Towards Automatic Controller Design using Multi-Objective Evolutionary Algorithms

Pedersen, Gerulf

Publication date:
2005

Document Version
Publisher's PDF, also known as Version of record

Towards Automatic Controller Design using Multi-Objective Evolutionary Algorithms

Ph.D. Thesis

Gerulf K. M. Pedersen

Department of Control Engineering
Aalborg University
Fredrik Bajers Vej 7C, DK-9220 Aalborg Ø, Denmark.

Dedication

To my father.

Acknowledgments

A large number of people have helped me in some way or the other since I started working on this thesis and until the final version at last was ready.

First of all, I would like to thank my family for the continuing support and encouragement they have given me, not only when I was growing up, but also within the last four years where I have conducted my PhD studies. I would especially like to thank my sister Michelle for accepting the difficult task of proofreading this thesis without having any knowledge of the subject whatsoever.

It is also prudent that I thank Hans-Christian Becker Jensen, Peter Fogh Odgaard, Zhuang Wu, and Brian Solberg all of whom I have shared offices with at some point during the last couple of years. It is not always easy having to put up with an officemate like me.

I would also like to thank the people of the Department of Control Engineering here at Aalborg University for their help and the many relevant and irrelevant discussions I have had with them over the last couple of years. A special thank goes out to the secretaries Jette Damkjær, Karen Drescher, and Jane Rasmussen Jamshidi for always giving me a helping hand with all the bureaucracy I had to endure over the years. Also, the technical expertise of Aage Nielsson, Ole Jørgensen, and Henrik Larsen has been greatly appreciated when it came to ensuring that the computers and network in the department were working properly.

A very special thanks goes to professor David E. Goldberg for letting me join his laboratory at University of Illinois at Urbana-Champaign in 2003. Without his expertise and that of IlliGAL, I would never have gotten such an insight into the field of evolutionary computation which I now have and which is the foundation for this thesis. In that connection, I would also like to thank all of the labbies whom I have had the pleasure of having countless discussions with over the years. These include Chang-Wook Ahn, Martin Butz, Jian-Hung Chen, Ying-Ping Chen, Elon Correa, Cristoph Domann, Nazan Khan, Pier-Luca Lanzi, Cláudio F. Lima, Xavier Llorà, Kei Onishi, Kumara Sastry, and Tian-Li Yu. I would also like to thank IlliGAL alumni Martin Pelikan for quite a number of interesting discussions as well.

During my stay at IlliGAL, there were quite a few administrative issues that needed to be resolved, and for helping me out with those I would like to thank Karen Czarnecki, Jeff

Leesman, and Jonathan Loveall working as part of the staff for the lab, but also Donna Eiskamp, Randall Elkins, and Debra Hilligoss working as staff for the Department of General Engineering.
Now, I would like to thank the members of my evaluation committee Jürgen Branke, Kalyanmoy Deb and Anders P. Ravn for accepting the responsibility of evaluating the research put forth in this thesis.
Finally, I would like to thank the two people who helped spark my interest in the field of evolutionary computation. These are, Anders Langballe, whom I wrote my master thesis with and who continually discusses evolutionary computation related issues with me, and also my supervisor Rafał Wiśniewski. Rafał, had it not been for your fascination of evolutionary computation, I would probably never have known about the subject and this thesis would never have been written.
Over the last four years, I have also discussed different subjects related to the work contained in this thesis with a variety of other people not mentioned here. However, despite the fact that I have not mentioned them in this acknowledgment does not mean that I am not grateful for those discussions and I offer my thanks to them as well.

Abstract

In order to design the controllers of tomorrow, a need has risen for tools that can aid in the design of these. A desire to use evolutionary computation as a tool to achieve that goal is what gave inspiration for the work contained in this thesis. After having studied the foundations of evolutionary computation, a choice was made to use multi-objective algorithms for the purpose of aiding in automatic controller design. More specifically, the choice was made to use the Non-dominated Sorting Genetic Algorithm II (NSGA-II), which is one of the most potent algorithms currently in use, as the foundation for achieving the desired goal.

While working with the algorithm, some issues arose which limited the use of the algorithm for unknown problems. These issues included the relative scale of the used fitness functions and the distribution of solutions on the optimal Pareto front. Some work has previously been done in this area using methods based on relative angles, utility functions, and projections and that work is what is extended in this thesis in order to cover a wider range of problems. This allows the NSGA-II to be transformed into a "black-box" optimization tool, which can be used for automatic controller design.

However, because the field of evolutionary computation is relatively unknown in the field of control engineering, this thesis also includes a comprehensive introduction to the basic field of evolutionary computation as well as a description of how the field has previously been used for solving a variety of issues in control engineering.

Synopsis

For at kunne konstruere morgendagens kontrolsystemer er der opstået et behov for værktøjer, der kan hjælpe med til at konstruere disse. Et ønske om at bruge evolutionære beregningsmetoder som et værktøj til at opnå dette mål er det inspirationsgrundlag som arbejdet med denne afhandling er baseret på. Efter at have studeret de fundamentale dele af evolutionære beregningsmetoder blev det valgt at benytte de multiobjektive algoritmer med det formål for øje at kunne hjælpe med til at automatisere konstruktionen af kontrolsystemer. Mere specifikt blev det besluttet at benytte "Non-dominated Sorting Genetic Algorithm II (NSGA-II)", som er en af de mest potente algoritmer af dem som benyttes i dag, som grundlag at at opnå det givne mål.
Under arbejdet med den givne algoritme fremkom der nogle emner som kunne begrænse brugen af algoritmen for ukendte problemstillinger. Disse emner inkluderede den relative skalering af "fitness" funktioner samt distribueringen af løsninger langs den Pareto optimale front. Der har tidligere været arbjede indenfor det felt som har baseret sig på relative vinkler, preferencebaserede funtioner samt projektioner og det er disse ting som bliver videreført i denne afhandling for at kunne dække et større sæt af problemer. Dette medfører at NSGA-II kan blive konvereteret til et "black-box" optimeringsværktøj som kan blive brugt til automatisk at kunne konstruere kontrolsystemer.
Fordi evolutionære beregningsmetoder er et relativt ukendt emne indenfor feltet der beskæftiger sig med kontrolsystemer vil denne afhandling også inkludere en meget dækkende introduktion til de mest basale dele af området. Derudover vil der også blive givet en beskrivelse af hvordan evolutionære beregningsmetoder tidligere har været brugt til at løse mange forskellige problemstillinger relateret til kontrolsystemer.

Contents

Nomenclature

Acronyms

AI	Artificial Intelligence.
ARMA	Auto Regressive Moving Average.
ARX	Auto-Regressive eXogeneous.
BB	Building Block.
BLX	Blended Crossover.
BOA	Bayesian Optimization Algorithm.
EA	Evolutionary Algorithm.
EC	Evolutionary Computation.
EP	Evolutionary Programming.
ES	Evolution Strategy.
FLC	Fuzzy Logic Controller.
GA	Genetic Algorithm.
GP	Genetic Programming.
hBOA	Hierarchical Bayesian Optimization Algorithm.
IA	Immune Algorithm.
LCS	Learning Classifier System.
LLGA	Linkage Learning Genetic Algorithm.
μGA	Micro Genetic Algorithm.
MOEA	Multi-Objective Evolutionary Algorithm.
MOGA	Multiple Objective Genetic Algorithm.
NN	Neural Network.
NSGA-II	Non-dominated Sorting Genetic Algorithm II.

PAES	Pareto-Archived Evolutionary Strategy.
PESA	Pareto Envelope-based Selection Algorithm.
PID	Proportional Integral Derivative.
PID-D2	Proportionate Integral Derivative Second Derivative.
PIPE	Probabilistic Incremental Program Evolution.
PMBGA	Probabilistic Model Building Genetic Algorithm.
PSO	Particle Swarm Optimization.
SA	Simulated Annealing.
SBX	Simulated Binary Crossover.
SHC	Stochastic Hill Climbing.
SPEA	Strength Pareto Evolutionary Algorithm.
SPEA2	Strength Pareto Evolutionary Algorithm 2.
SUS	Stochastic Universal Sampling.

Symbols

Γ	Preferential operator.
Λ	Non-dominated operator.
A_r	Resulting set.
ι	Termination criterion.
λ	Number of offspring individuals.
μ	Number of parent individuals.
Ω	Set of genetic operators.
Φ	Mapping of fitness functions into $\mathbb{R}^M$.
Ψ	Generation transition function.
σ_{share}	Size of niche around an individual.
τ_{EA}	Running time.
Υ	Genotypic to phenotypic mapping.
A_t	Sub-optimal set at time t.
I	Space of individuals.
N_{d_i}	Number of individuals which dominate individual i.
N_{front}	Number of individuals belonging to a specific front.
N	Population size.
s	Selection operator.

List of Figures

List of Tables

Chapter 1

Introduction

Ever since engineers started using controllers and thus had to design them as well, they have also been concerned with the issues of how to find the optimal controller for a specific application or perhaps derive a more general controller for control of a certain class of systems. Much work is currently going into control theory and design, and the techniques used for handling those issues and the results obtained keep on improving. Some areas, however, continue to haunt control engineers because the optimal solution or design method cannot be found using currently known techniques. Some of these difficult areas include things such as non-linear control, optimization of parameters for a non-convex region, and unifying/recombining different controller structures to obtain better controllers. Another challenge for control engineers is to keep up with the fast development of different hardware, actuators, and sensors, since it provides more and more possibilities for design and control. As such, it is very difficult to find optimal controllers and controller structures for the ever expanding array of systems.

For a long time the field of control engineering has leaned greatly on the advances in areas such as mathematics, computer science, and mechanical engineering, which have helped in furthering the control engineering field considerably. However, no matter how much research have gone into control, it is not yet possible to automate the controller design procedures for advanced systems. The time might never come, but it would be a great step on the way if the design of complex controller structures and systems could be at least partially automated.

Being inspired by other fields of research and applying such theories and methods to ones own field/application is becoming more and more widespread and also necessary. In the same way as control has learned from related fields, so have those related fields gotten inspiration from other fields as well. One of the growing fields within the last decades have been the mutual inspiration of computer science and biology. In an attempt to make computers smarter and behave more intelligent, computer scientists have been inspired by the successful ways of nature. Similarly, biologists have been inspired by the simulation of such artificial evolution to improve their knowledge of biological interactions and behavioral patterns. The result of this interaction between computer science and biology has, among other things, spawned the promising field of evolutionary computation, which is the basic foundation that will be used in this thesis.

1.1 Background

The work contained in this thesis actually started when I was working on my Masters thesis together with Anders Langballe in the spring of 2001. The idea then was to use evolutionary computation for finding a mixed H_2/H_∞-controller and the work culminated in the Masters thesis Langballe and Pedersen (2001) and a conference paper (Pedersen, Langballe, & Wiśniewski, 2002). When putting the final touches on the Masters thesis, the idea of using evolutionary computation for automatic design of controllers was conceived. It is that idea, which is the basis for this thesis, and as will become apparent, the subject was not so straightforward as could have been desired.

At first, the approach to the subject was based in the field of control engineering. However, it soon became apparent that an extensive amount of knowledge in the field of evolutionary computation was necessary in order to achieve the goal set forth. Luckily, it was possible for me to join the renowned Illinois Genetic Algorithms Laboratory (IlliGAL) in 2003 where I acquired an extensive amount of knowledge in the field. It is this knowledge that has allowed me to conduct the research presented in this thesis and hopefully it will be presented in such a way that it will reflect on to the readers of this thesis as well.

1.2 Contributions

The main purpose of this thesis is to make a series of contributions to the field of evolutionary computation such that it in time will become possible to use those methods for automatic design of controllers. However, this thesis is also meant as an introduction to the field of evolutionary computation for the many control engineers who have not yet realized the major potential that lies in using evolutionary computation for design of controllers. This means that even though some of the issues presented in this thesis are known in the field of evolutionary computation, it is not at all common knowledge in the field of control engineering. The contributions of this thesis are thus given as:

Compact introduction to the field of evolutionary computation In order for control engineers to become aware of the field of evolutionary computation, a brief but comprehensive introduction is given to the different areas that fall under the field of evolutionary computation. In connection with this, an existing general framework for single-objective evolutionary algorithms is expanded to also cover the field of multi-objective optimization as well.

Comprehensive survey of evolutionary computation in control This thesis also includes a comprehensive survey of the vast number of areas within the field of control engineering where evolutionary algorithms already have been applied in one way or another. This is to give an overview of, but also inspiration to, the different ways that evolutionary computation have been and can be used for different control related purposes.

Resolve issues caused by unknown scalings of fitness functions When it comes to using multi-objective algorithms, it turns out that the algorithm used in this thesis had problems

when it encountered disparate scalings of the fitness functions used in the optimization process. This issue is addressed and different solutions to the problem, which should more or less be capable of resolving the issue, are presented.

Improve the distribution of solutions when using multi-objective evolutionary computation For the multi-objective case where several different fitness functions are used in the optimization process, the result is usually a set of different solutions that all can be considered optimal. However, the distribution of those solutions is found to be non-optimal, since some of the solutions can be considered to be more trivial than others. Several different ways to resolve this issue and obtain a set of solutions, which best represent the set of non-trivial optimal solutions, are proposed and the method which is best at obtaining the desired distribution of solutions among the proposed methods is identified.

First step to a black box optimizer The last issue of this thesis that is considered as a contribution is some preliminary steps that can be applied to make the algorithm used throughout this thesis independent of the problem being solved. By separating the algorithm from the problem, the algorithm will be easier to use for solving a wide variety of different problems which may require the use of special programs in order for the fitness to be calculated. To limit the need for expert knowledge in connection with using the algorithm, a set of recommended parameters is also presented that is expected to yield good results for most problems.

1.3 Overview of Thesis

Based on the contributions mentioned in the previous section, it is possible to give a more exact overview of the structure of this thesis.

Chapter 2 In this chapter, the introduction to the field of evolutionary computation is given at such a level that no prerequisite of the field is necessary. The different areas which make up the field of evolutionary computation will be discussed and a general framework that includes the field of multi-objective optimization is presented here.

Chapter 3 A comprehensive survey to the use of evolutionary computation within the field of control engineering is given in this chapter. The survey covers a wide variety of control related areas and also makes connections to the fields of fuzzy logic and neural networks.

Chapter 4 A discussion of fitness functions and objectives are presented in this chapter. When it comes to using evolutionary computation for solving different problems, the issue that always must be addressed is how the fitness functions and possible constraints should be formulated. The differences between using either multiple or just a single fitness function are discussed along with the influence the use of constraints might have on the evolutionary process.

Chapter 5 Here, the problems that might be encountered when using the NSGA-II algorithm with fitness functions that have disparate scaling are presented and resolved. Because of the popularity of the NSGA-II algorithm, it is the algorithm which is used here and in the remaining chapters of this thesis.

Chapter 6 For the case involving two fitness functions, a series of four different methods for obtaining a good distribution of the set of non-trivial solutions is presented. The methods are based on estimated derivatives, angles, and circumradius and they are tested on a variety of problems followed by a discussion of the advantages and drawbacks of each method.

Chapter 7 In this chapter, another approach to obtaining a good distribution of the set of non-trivial solutions is investigated. This method is based on projections and is applicable to cases of any dimension. The approach is tested on the same set of problems used in the previous chapter.

Chapter 8 This chapter is concerned with a way of separating the calculation of the fitness functions from the main NSGA-II algorithm. This is attempted in order to expand the use of NSGA-II to also cover cases that require the use of very specific programs or methods for calculation of the fitness functions. Further, a set of recommended parameters for NSGA-II that is expected to yield good results for any given problem is presented.

Chapter 9 The conclusion of this thesis will discuss the issues presented throughout this thesis and will also give some perspectives on some further work that will help in achieving the goal of using multi-objective evolutionary algorithms for automatic design of controllers.

Having given this outlook of what is to come, let us first begin with a fundamental description of what evolutionary computation really is.

Chapter 2

Evolutionary Computation

This chapter is concerned with giving the reader some basic knowledge of Evolutionary Computation (EC). The name itself comes from the desire of using **evolutionary** methods for solving **computational** difficult problems. Evolutionary computational methods are a way to perform numerical optimization based on the Darwinian theory of evolution. As such, it is a simulated version of natural selection, also commonly known as "survival of the fittest". The first part of this chapter will give an introduction to the notion of Darwinian evolution followed by a description of the major stages of the evolutionary process. These stages are then used as a basis for a general mathematical description of an Evolutionary Algorithm (EA). This generalized expression will then be used to show how the different areas of Genetic Algorithms, Evolution Strategies, Evolutionary Programming and Genetic Programming are related. First, some basic knowledge of evolution is needed and as such the background of the evolutionary principle will be presented.

2.1 Background

To begin with, it is advantageous to get a better understanding of the underlying elements of the Darwinian theory of evolution. The basic foundation of the Darwinian theory is that when changes happen to an environment, the individuals who are better suited for the new environment will prosper more than those who are less suited for the environment. As such, for a population of individuals, those who are better fit for the environment will guide the evolution of their species in a direction that is better suited for the environment, whereas the less fit individuals will die out. The result will be a population that continually will adapt to the environment provided that the environment does not change too radically too fast.

2.1.1 Evolution and Natural Selection

Part of what inspired the Darwinian theory of evolution is the case that bears his name, Darwin's finch. When Darwin first visited the Galapagos Isles, he collected many samples of animal species,

including birds. At first, he thought many of the different birds were of different species, but when he got back from his travels he was told by John Gould that the birds were all finches, and as such the collection of 14 different species of finches are called Darwin's finch. He then went along and formulated his theory of evolution as an explanation to how these different species of related finches could have come to be (Johnson, 2004). Since Darwin formulated his theory of evolution, there have been a considerable amount of studies related to the finches. One study took place over a period of 30 years and some of the results from the study confirmed the theory of natural selection which is part of Darwins theory (Grant & Grant, 2002). The study showed that during a drought where many of the plants producing small seeds were eliminated, the finches with large beaks did not suffer as great a reduction in numbers as those with small beaks. The finches with small beaks were unable to crack open the larger remaining seeds and thus reduced drastically in numbers. However, when a drought affected the plants producing large seeds, then the small beaked finches did not suffer as much of a reduction as the large beaked finches. The large beaked finches were unable to handle the small seeds with their large beaks and thus their numbers reduced drastically. One of the finch species did not suffer greatly during either drought because the diet for this species depended less on seeds than the rest of the finch species. This clearly shows how the species that are most adapted to an environment are better at surviving than those who are not, and thus confirms this part of Darwins theory of evolution.

This notion of evolution is a very powerful tool exploited in nature for adaptation of different species for a variety of changing environments. It is not surprising that evolution has inspired computer scientists in such a way that the strength of the natural evolution could be applied to the tasks of creating Artificial Intelligence (AI) and intelligent adaptive problem solvers. The result was the emergence of several related research areas inspired by evolution. The next part of this thesis will focus on some of the basic knowledge in biology. By understanding biology and the terminology used, it will be much easier to better understand how and why evolutionary computation works. Following that a general description of the basic evolutionary computation structure will be given. This will be done because many of the research areas with origin in evolution have similar traits. From these common features it is thus possible afterwards to obtain the different evolutionary computation methods.

2.2 Biological Foundations

Here on Earth the basic building blocks of naturally evolved life are comprised of genes coded as a sequence of DNA. A gene is a recipe for a specific feature for whatever life form it is present in and the length of a gene can vary considerably in size. For each gene several alternatives called alleles can exist. As such, a gene which determines eye color for a human has several alleles signifying the colors blue, green, brown, or a combination of those. In order to keep the genes organized and compact nature devised the use of chromosomes. Thus, a chromosome is a tightly packed collection of genes. Depending on the species of an individual, it will consist of one or more chromosomes.

When looking at genetics, it can be viewed from two different angles. It can either be looked upon as a recipe or as a finished product. In this case, the recipe is the genes and the finished product is the interpretation of said genes, the phenotype. When a change occurs in a gene, a corresponding change might be seen in the phenotype depending on the interpretation used. With these few terms

in place, it is now possible to continue with the basic structure of the evolutionary computation approaches.

2.3 Basic Evolutionary Computation Structure

When looking at evolutionary computation as a whole, the different methods used all have common features inspired by the theory of evolution. This can be considered as the basic foundation on which different methods and abstraction levels can be used to solve computational difficult problems. The structure of such an evolutionary approach consists of a population base which repeatedly undergoes:

- Evaluation
- Selection
- Alteration

The structure of this basic evolutionary process is also shown in figure 2.1. Each iteration of

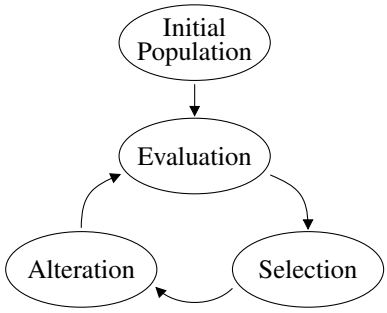


Figure 2.1: Overall structure of basic evolution.

this sequence corresponds to the creation of a new generation based on the population of the previous generation. The result is thus a population that continuously adapts to the environment it is exposed to. The main requirement for using such an evolutionary process is that it must be possible to evaluate how well an individual performs in the environment.

2.3.1 Evaluation

Evaluation of the individuals in the population determines how fit each individual is at coping with a given environment, which is basically a mapping from a phenotype to one or more objective/fitness values. As such, it gives a preferential measure that can be used to bias the population in a given direction. Depending on the environment, there can be several traits that are desirable for the individuals to have. This is easily realized using the following example of herbivores:

Example 2.1 (Herbivores) *In nature, an herbivore needs to be able to find food while also being able to either avoid or defend against hungry carnivores in order to ensure survival. The ability of finding food is not sufficient, since the appearance of a carnivore could be fatal to the herbivore. Accordingly, the ability to avoid being killed by a carnivore is not sufficient without the ability to find food, as the herbivore would starve to death. If no carnivores had been present in the environment, it would suffice for the herbivores to be able to find food in order to ensure survival.*

The example clearly illustrates how the desired traits of the individuals depend primarily on the environment. Those who are better fit for survival in an environment, will have a higher chance at continuing survival. Based on this fitness, the individuals are selected for inclusion in the next generation.

2.3.2 Selection

The evaluation of the individuals is performed such that it can be decided to which extent different individuals should survive. In this "survival of the fittest" approach some individuals with very good fitness would in general be preferred to individuals with low fitness, but also some cases lower ranked individuals could be lucky and survive. If a few individuals with low fitness are capable of surviving, they allow for a continued exploration of the search space for more fruitful regions. An example of this kind of selection can be seen from Darwin's finch:

Example 2.2 (Darwin's finch) *Droughts that affect plants producing small seeds will tend to favor those finches with large beaks, since the finches with smaller beaks have more difficulty handling the large seeds. However, even though the smaller beaked finches are less fit than the finches with large beaks, some of them manage to survive. As a result, the finches with small beaks that survive the drought can replenish their numbers when the drought has ended. In case a drought then affects the plants producing large seeds, the finches with smaller beaks will be more fit for the environment than finches with larger beaks. Thus, had the lesser fit finches not survived the drought, then the droughts could most likely have eradicated both the small and large beaked finches.*

The example above describes how changes in the environment affected the population. For a fixed environment, the same considerations have merit since one area of the environment might be unsuitable for some individuals, whereas other parts of the environment would be more suitable. In the above example, this would correspond to a migration of the affected finches to another area not affected by droughts.

Selection on its own will not allow the full potential of evolution to occur. If only selection took place, the result would be that the best individual of the initial population, which may be far from optimal, would quickly dominate the entire population. As such, selection corresponds to an exploitation of the existing individuals. However, in order to evolve there is a need for discovery or exploration as well. The exploration will allow for new individuals with different features to emerge, which may or may not prove better than those already present in the population. This makes it a necessity to perform some alterations on the population in order to explore for new and better features for the individuals.

2.3.3 Alteration

In order to perform a search on the search space, it is necessary to make some alterations to some of the surviving individuals. With the genetics in mind, the individuals with higher than average fitness must have had features that were superior either individually or in conjunction with other features. In nature, the features of an individual is contained in the genome. Since the genes interact with each other to create the different features, the genes themselves can be considered as Building Blocks (BBs) (Goldberg, 2002). These building blocks can be recombined to form new and perhaps better features. Based on the fact that superior individuals must have had features that were superior, the building blocks of those individuals, must have been better than average. Recombining these better than average building blocks should thus on average yield increasingly better features in the resulting individuals. This can also be realized from the following example:

Example 2.3 (Recombination) *In a group of herbivores, features such as primitive sight and/or smell would make the individuals having these features better fit for finding food and avoiding predators than those without such abilities. If one of these herbivores with only primitive sight were to mate with an herbivore with a primitive sense of smell, they could potentially create an offspring with both those features which would make it more fit than either of the parents.*

Recombination is, however, not the only alteration that needs to be performed on the individuals. This would only lead to an exploitation of building blocks already present in the initial population and not allow for new discoveries to take place. The issue can be solved by introducing mutations. The advantage and the drawback of mutations is the unpredictability. Even though mutations can create new or improve on existing good features, they can also result in destruction of useful features or create unwanted features.

Example 2.4 (Mutation) *For an animal living in a dry environment, such as a desert, a mutation that will allow an individual to conserve water better or more efficiently would help in aiding the chances of survival. Counter to that is the case where a mutation could require the individual to use more water or at a faster rate, which would render the individual more vulnerable to the scarce presence of water in the dry environment.*

Using recombination and mutation to alter the individuals in each successive generation, it is with the aid of evaluation and selection, possible to successfully evolve useful and non-trivial solutions to a wide set of optimization problems. Since the basic understanding of the evolutionary computation structure should now be known to the reader, we can proceed to a more technical description of these evolutionary based algorithms.

2.4 General Evolutionary Algorithm

In Bäck (1996), a concise description of a general single-objective evolutionary algorithm framework is given. A total of four definitions are given in order to describe the structure of an algorithm, the transitions between generations, running time, results, and the operators. Those

definitions are so general that the definitions in this thesis will be based on these. Some modifications will, however, be necessary because Bäck (1996) only describes algorithms having one fitness value whereas this thesis also covers algorithms with multiple fitness values.

Let us first define a general evolutionary algorithm:

Definition 2.4.1 (General Evolutionary Algorithm) *An **Evolutionary Algorithm (EA)** is defined as an 8-tuple*

$$\boldsymbol{EA} = (I, \Phi, \Omega, \Psi, s, \iota, \mu, \lambda) \tag{2.1}$$

*where $I = A_x \times A_o$ is the space of **individuals**, and A_x, A_o denote arbitrary sets. $\Phi : I \to \mathbb{R}^M$ is a mapping from the space of individuals to the M dimensional space of real values ($M \in \mathbb{N}^+$). The mapping Φ consists of M **fitness functions** $f_1, \ldots, f_M$ which each assigns a real value to the individuals ($f_i : I \to \mathbb{R}, \forall i = 1, \ldots, M$).*

*μ is the **number of parent individuals**, while λ denotes the **number of offspring individuals**.*

*Using the shorthand notation $\omega_{\Theta_i} : I^x \to I^y$ for a corresponding mapping $\omega_i : \Theta_i \times I^x \to I^y$, the following description for the set of probabilistic **genetic operators** Ω can be obtained.*

$$\Omega = \left\{ \omega_{\Theta_1}, \ldots, \omega_{\Theta_z} | \omega_{\Theta_i} : I^\lambda \to I^\lambda \right\} \cup \left\{ \omega_{\Theta_0} : I^\mu \to I^\lambda \right\} \quad , \tag{2.2}$$

where each ω_{Θ_i} represents a genetic operator which is controlled by specific parameters summarized in a corresponding set $\Theta_i \subset \mathbb{R}$.

*The **selection operator** is denoted by s. The parameter set for s is denoted by Θ_s, but there is one specific element of Θ_s, namely $\theta_{s_{el}}$ that has a major influence on the selection operator. The element $\theta_{s_{el}}$ can be either $true$ or $false$ depending on whether the algorithm uses elitism or not. The mapping of the selection operator is then given by*

$$s : \begin{cases} \Theta_s \times I^{\mu+\lambda} \to I^\mu & \text{for } \theta_{s_{el}} = true \\ \Theta_s \times I^\lambda \to I^\mu & \text{for } \theta_{s_{el}} = false \end{cases} . \tag{2.3}$$

In fact, selection changes the number of individuals from λ or $\lambda + \mu$ to μ, depending on the value of $\theta_{s_{el}}$.

*Finally, $\iota : I^\mu \to \{true, false\}$ is a **termination criterion** for the **EA**, and the **generation transition function** $\Psi : I^\mu \to I^\mu$ describes the complete process of transforming a population $P(t)$ into a subsequent one by applying genetic operators and selection:*

$$\Psi(P(t)) = s(\theta_s, \Phi, P(t), \omega_{\Theta_1}(\theta_1, \ldots(\omega_{\Theta_z}(\theta_z, \omega_{\Theta_0}(\theta_0, P(t)))))) \quad , \tag{2.4}$$

where the elements θ_i belong to the corresponding set Θ_i, θ_s belongs to Θ_s, $P(t) \subset I^\mu$ is an instance of the population at time t, and Φ is the mapping of each individual given by

$$\Phi : \begin{cases} I^{\mu+\lambda} \to \mathbb{R}^M & \text{for } \theta_{s_{el}} = true \\ I^\lambda \to \mathbb{R}^M & \text{for } \theta_{s_{el}} = false \end{cases} . \tag{2.5}$$

For the space $I = A_x \times A_o$, the sets A_x and A_o can be arbitrarily complex. A_x denotes a set representing the problem specific part of the individuals, whereas the set A_o represents the

operator specific information contained in the individuals (see figure 2.2). As such, the A_o part of the individuals is the term which allows the parameters of the EA to be continually updated, thus making the EA self-adaptive.

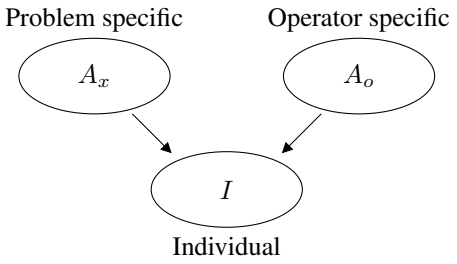


Figure 2.2: The structure of an individual in the general evolutionary algorithm framework.

The fitness functions given in Φ are based on the number of objectives and the number of constraints. In some cases, constraints are included as a penalizing term in one or more of the fitness functions, resulting in a Φ consisting solely of objective functions. In other circumstances, the constraints can be considered as separate fitness functions thus contributing to additional dimensions of Φ. The dimension M thus either corresponds to the number of objectives or the combined number of objectives and constraints depending on the implementation.

The operator ω_{Θ_0} is used to create a mating pool of size λ from the μ parent individuals. This is usually only done for the cases when $\mu \neq \lambda$, but there are some algorithms which specifically use this operator for creation of a mating pool even for the case when $\mu = \lambda$.

Once the offspring have been subjected to the remaining genetic operators $\omega_{\Theta_1}, \ldots, \omega_{\Theta_z}$, the selection mechanism is applied such that the μ individuals of the next generation can be found. For the case when elitism is not used $\theta_{s_{el}} = false$, only the λ offspring is used to find the μ individuals of the next generation. If, on the other hand, some degree of elitism is used $\theta_{s_{el}} = true$, a union of offspring and their parents $I^{\mu+\lambda}$ is used in the selection operator to find the μ individuals of the following generation. The genetic operators are stochastically based on exogenous input, whereas the selection operator can be either stochastically or deterministically based. An illustration of how the genetic operators and selection are applied to the general EA framework is shown in figure 2.3 on the following page.

The termination criterion can be based on exogenous criterions such as a certain limit to the number of generations, but it can also be based on internal factors of the algorithms like averages or densities.

The generation transition function Ψ corresponds to the completion of one generation, which is comprised of the application of the genetic operators followed by selection. Using this generation transition operator Ψ, it is thus possible to define a population sequence:

Definition 2.4.2 (Population sequence) *Given an Evolutionary Algorithm with generation transition function* $\Psi : I^\mu \rightarrow I^\mu$ *and an initial population* $P(0) \in I^\mu$*, the sequence*

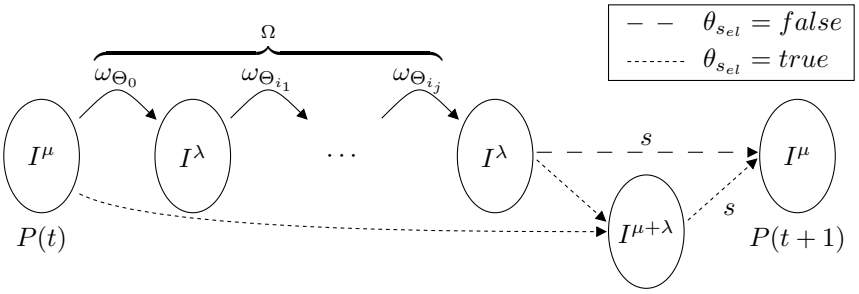


Figure 2.3: Transition from one generation, $P(t)$, to the next, $P(t+1)$, by application of genetic operators and selection to the general evolutionary algorithm framework. The genetic operators are always applied in a similar manner, but the selection depends on the value of $\theta_{s_{el}}$ being either $false$ (long strokes) or $true$ (short strokes).

$P(0), P(1), P(2), \ldots$ *is called a* ***population sequence*** *or* ***evolution*** *of* $P(0)$*:* $\Longleftrightarrow$

$$\forall t \geq 0 \quad : \quad P(t+1) = \Psi(P(t)) \quad . \tag{2.6}$$

Initialization of the algorithm for $P(0)$ can either be done randomly, which is the usual case, or based on one or more regions, where the optima are known/assumed to be. The population sequence then runs continually until the termination criterion have been reached. For the cases when the population sequence turn out to be convergent, the EA would be well defined in a mathematical sense. With the definition of a population sequence in place this naturally leads to the definition of the running time.

Definition 2.4.3 (Running time) *Given an initial population* $P(0) \in I^\mu$ *for an EA with generational transition function* Ψ*, the running time* τ_{EA} *is given by*

$$\tau_{EA} = \min\left\{t \in \mathbb{N} | \iota\left(\Psi^t(P(0))\right) = true\right\} \quad . \tag{2.7}$$

The relationship between a population sequence and the running time is illustrated in figure 2.4 on the facing page. The figure also relates the termination criterion with the resulting set A_r. Based on the running time of the algorithm, it is possible to define the resulting set in the following:

Definition 2.4.4 (Resulting Set) *For each generation* $t = 0, \ldots, \tau_{EA}$ *an operator* Λ *can be applied, which selects only those individuals that are* ***non-dominated***[1] *with respect to the other individuals of that particular generation. This results in the suboptimal sets*

$$A_t = \Lambda\left(\Psi^t(P(0))\right) \qquad \forall t = 0, \ldots, \tau_{EA} \quad . \tag{2.8}$$

[1] A definition of this is given in definition 2.10.3 on page 44

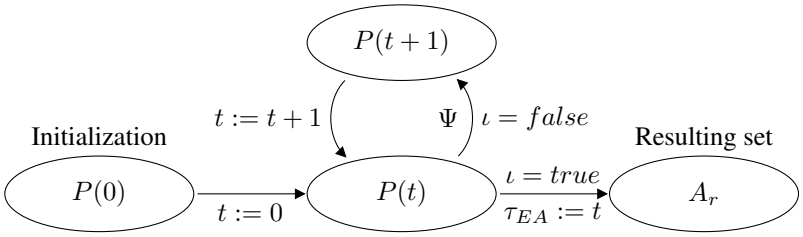


Figure 2.4: Population sequence with termination criterion. Assignments are indicated with ':=' whereas regular expressions are indicated with '='.

Taking the union over all t *and applying a user based preferential operator* Γ *it is then possible to obtain the* ***resulting set*** *as:*

$$A_r = \Gamma\left(\bigcup_{t=0}^{\tau_{EA}} A_t\right) \tag{2.9}$$

The resulting set is sometimes also denoted as the ***result*** *of an EA.*

In figure 2.5, it is illustrated how the resulting set is found based on the evolutionary process.

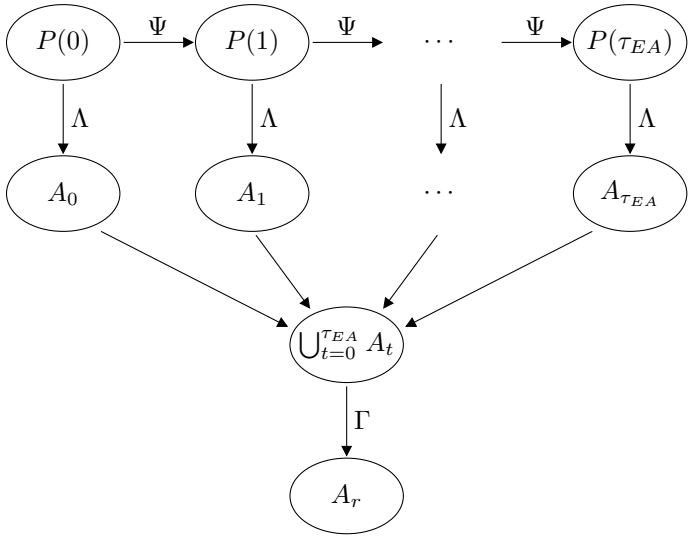


Figure 2.5: Resulting set based on the individuals of the entire evolutionary process.

For an algorithm with only one fitness function, a single-objective algorithm, and where a single optimum value is sought, the preferential operator Γ will usually be given such that the resulting set A_r will have a size of one, namely the optimum value. This means that Γ will select a single

optimum value among the sub-optimal sets A_t. Referring to figure 2.5 on the preceding page, the sub-optimal sets $A_0, A_1, \ldots, A_{\tau_{EA}}$ will consist of the best values at each time t and the resulting set A_r will be chosen as the optimum value amongst these sub-optimal sets.

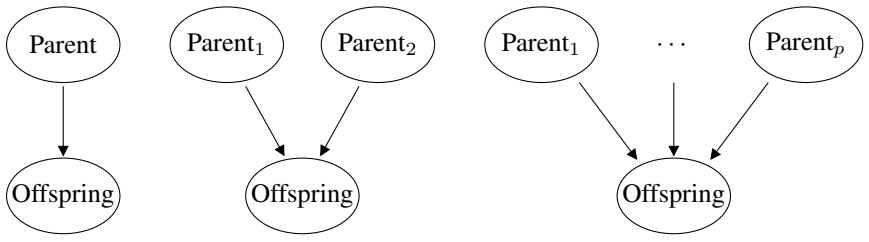
For algorithms with more than one fitness function, the preferential operator Γ and the non-dominated operator Λ can be chosen in a variety of ways, which will be further explained in section 2.10.4 on page 52.

Before continuing with the more detailed description of the different variations of evolutionary computation, let us take a closer look at a general description of the genetic operators. What have been described in the earlier definitions was high-level descriptions which transformed an entire population into another population. The following definition will give a low-level description of how the genetic operators are used to create each individual.

Definition 2.4.5 (Asexual, sexual, panmictic genetic operators) *A genetic operator $\omega_\Theta : I^p \to I^q$ is called:*

$$
\begin{array}{lll}
\textit{asexual} & :\Leftrightarrow & \exists \omega'_\Theta : I \to I : \\
 & & \omega_\Theta\left(\vec{a}_1, \ldots, \vec{a}_p\right) = \left(\omega'_\Theta\left(\vec{a}_1\right), \ldots, \omega'_\Theta\left(\vec{a}_p\right)\right) \wedge p = q \ , \\
\textit{sexual} & :\Leftrightarrow & \exists \omega'_\Theta : I^2 \to I : \\
 & & \omega_\Theta\left(\vec{a}_1, \ldots, \vec{a}_p\right) = \left(\omega'_\Theta\left(\vec{a}_{i_1}, \vec{a}_{j_1}\right), \ldots, \omega'_\Theta\left(\vec{a}_{i_q}, \vec{a}_{j_q}\right)\right) \\
 & & where \ \forall k \in \{1, \ldots, q\} \ \ i_k, j_k \in \{1, \ldots, p\} \\
 & & \ \ are \ chosen \ at \ random, \\
\textit{panmictic} & :\Leftrightarrow & \exists \omega'_\Theta : I^p \to I : \\
 & & \omega_\Theta\left(\vec{a}_1, \ldots, \vec{a}_p\right) = \Big(\underbrace{\omega'_\Theta\left(\vec{a}_1, \ldots, \vec{a}_p\right), \ldots, \omega'_\Theta\left(\vec{a}_1, \ldots, \vec{a}_p\right)}_{q}\Big)
\end{array}
\tag{2.10}
$$

An illustration of how the different genetic operators are used to create an offspring from a set of parents can be seen in figure 2.6. This process is then done q times such that the offspring population can be filled.



(a) Asexual. (b) Sexual. (c) Panmictic.

Figure 2.6: The different genetic operators.

Let us shortly compare the genetic operators with the biological equivalents. The asexual genetic operator is equivalent to mutation and self replication. With regard to self replication, this method of reproduction can be found in nature in low level life forms such as amoebas and bacteria. The mutation part of the operators comes to show since it is not always possible to make exact copies in nature and as such, the replicas will not always be identical to the original.

The sexual genetic operator corresponds to the well-known reproduction method of most higher level life forms. Two distinct individuals produce an offspring that is a recombination of the genetic material of both parents. A noticeable difference from the sexual reproduction of nature to the one explained in definition 2.4.5 on the facing page is the fact that no distinct sexes are required. Whereas nature requires a male and female for sexual reproduction to take place, the definition given have no such requirement.

Finally, we have the panmictic genetic operator, which have no biological equivalent. It is a purely artificial reproduction operator which allows for the recombination of more than two parents when creating an offspring. In principle, the panmictic genetic operator could be written as a finite number of sexual genetic operations, but the usefulness of this approach would be limited because of the hierarchical structure in which this approach would have to be performed and because of the bias that will be accumulated in the resulting offspring.

The definitions given in this section have given a general framework upon which most evolutionary computation methods is based. In the next couple of sections, those definitions will be specialized somewhat to show how the different evolutionary based computation methods can be derived. However, the different EAs are not limited to the details described in the following sections, since many different and specialized versions of algorithms and operators exist. The following descriptions are thus based on general and in some cases fundamental properties of the algorithms belonging to the different classes.

2.5 Genetic Algorithms

Genetic Algorithms (GAs) in the form known today was first proposed by John Holland in the early 1970s. His book "Adaptation in Natural and Artificial Systems" (Holland, 1975) is considered to have laid the foundation upon which the GAs of today have been built. It was, however, not until Goldberg published his book "Genetic Algorithms in Search, Optimization, and Machine Learning" (Goldberg, 1989) that the field of GAs really started to evolve.

Based on the general EA framework given previously, the specifics for how a GA can be expressed will be presented in the following. For the case of the real valued GAs, the equations behind the different operations are not shown, but can be found in Deb (2001). That book has also been used as inspiration for some of the figures relating to real valued GAs.

2.5.1 Representation

In GAs, the representation of the individuals, also known as encoding, consists of a set of genes. Originally, these genes were represented using binary strings (Holland, 1975; Goldberg, 2002), but later on the use of real values have also become popular, inspired by the success of real valued encodings in Evolution Strategies (see section 2.7 on page 33). There exist a connection between

binary and real valued representations, since binary values can be mapped into real values and vice versa. This is not saying that the representations are equal because the mapping between such representations imposes an additional structure to a given problem and the difficulty of many problems can actually be seen to arise from the use of inappropriate representations.

The genes can be viewed as a blueprint for an individual, as in the following example:

Example 2.5 (Construction of a Car) *When manufacturing a car, it is first necessary to have some measurements or blueprints to construct the car from. As such, the blueprints can be viewed as the genome of the car and the car itself can be viewed as the individual. Simply put, the blueprints (genotype) of the car is a representation (or recipe) of how to make the actual car (phenotype).*

As seen in the above example, the mapping from the genotype to the phenotype when using binary encoding is simply a matter of constructing the individual based on the genetic structure of that individual. A simple example of such a conversion from genotypic space to phenotypic space can be seen in figure 2.7 using the syntax of Bäck (1996) where Υ indicates a genotypic to phenotypic mapping.

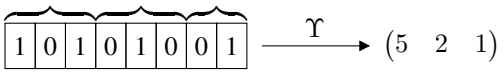


Figure 2.7: Mapping from binary genotype to real valued phenotype.

For a real valued encoding such a mapping is unnecessary since the phenotype is directly given in the individual. All of the individuals, binary or real valued, are contained in the set I, and the representation of that set corresponds to the elements of A_x. If some operator specific knowledge is incorporated into the individuals, that information is included in the individuals through the set A_o either using binary or real values. It is, however, very few GAs which include such operator specific knowledge in the individuals.

Having the genotypic and phenotypic representation of the individuals given, it is then a matter of assigning fitness values to them.

2.5.2 Evaluation

Evaluation is the way the individuals are assigned fitness values according to how well they perform in a desired context or environment. This can best be explained by following up on example 2.5:

Example 2.6 (Car Performance) *How well a car performs, depends on the context. A racing car optimized for driving fast on flat roads does not perform well when having to drive off-road. Similarly, an off-road car cannot be expected to outperform a racing car on flat roads.*

The evaluation is thus a mapping from the phenotype of an individual to the fitness space determined by the context. As such, the assignment of fitness values to the individuals corresponds to the general EA mapping of Φ from I into $\mathbb{R}^M$.

For a regular GA, the fitness space is usually one dimensional, which means that the fitness is given by assigning each individual a single fitness value according to how well it performs with regard to the given context. Relating this to the general EA description from section 2.1 on page 10 it corresponds to $M = 1$. A discussion of what is done for $M \neq 1$ is given in section 2.10 on page 41.

The evaluation gives a measure on how to distinguish good individuals from inferior ones, but it is also necessary to have a scheme for choosing the most promising individuals for continued survival when evolving. Some of these selection schemes will be discussed next.

2.5.3 Selection

Selection, denoted as s in the general EA framework, is the process that decides whether an individual survives to reproduce or dies out. As such, selection can be considered as the survival part of "survival of the fittest". The selection will primarily select individuals with higher fitness values for survival, but with some selection schemes it is also possible for individuals having lower fitness values to survive to some extent. This will cause the population to gradually improve over the span of evolution, but the selection of some less fit individuals will also allow for diversity and a chance of discovering other promising parts of the population space.

For regular genetic algorithms selection is performed according to a mathematically calculated fitness function, an objective function. However, in some cases selection can also be performed by a human, the so-called human based GAs, and the fitness function used in the selection process is thus determined using a subjective evaluation, a subjective function.

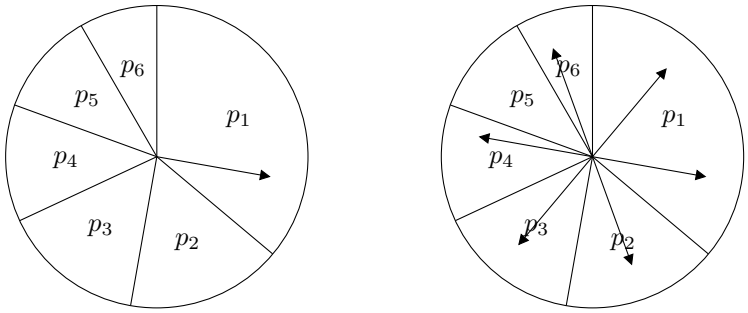
For the objective case, there exist several different selection methods. Some are based on proportionate selection, which uses the proportion of each individuals fitness value compared to the average fitness value of the entire population to decide the possibility for selection. One restraint for this kind of selection is, however, that the fitness values must be positive, and if negative values were to occur, a mapping must be introduced that will transform the fitness values into the positive half plane $\mathbb{R}^+$. Some proportionate based selection strategies are roulette selection and Stochastic Universal Sampling (SUS). An example of both roulette selection and SUS can be seen in figure 2.8 on the next page. For roulette selection, the sampling procedure chooses one individual in each trial and is repeated μ times, such that the population for the next generation of parents can be filled. In the case of SUS, the roulette wheel is equipped with μ equispaced arrows, such that the sampling is performed only once when filling the population for the next generation.

Another stochastic based selection method is tournament selection, which selects a number of individuals randomly and among those the best individual is selected, based on a comparison of the fitness values. An illustration of the principle of tournament selection can be seen in figure 2.9 on the following page.

There also exist deterministic selection methods such as truncation selection. Truncation selection selects a fraction of the top most fit individuals and copies them to fill out the rest of the population, thus eliminating all other individuals. This is illustrated in figure 2.10 on the next page.

The first regular GAs usually used a (λ, μ) selection scheme[2] where the offspring individuals, λ, completely replaces the parent individuals, μ, for the next generation and where $\mu = \lambda$ (see

[2]The terminology is further explained in section 2.7 on page 33



(a) Roulette selection.

(b) Stochastic universal sampling.

Figure 2.8: Proportionate selection on a population consisting of 6 individuals.

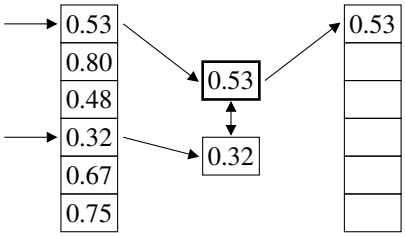


Figure 2.9: Tournament selection with size 2 on a maximization problem.

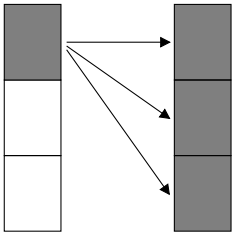


Figure 2.10: Truncation selection with size 3.

the case with the long stroked lines in figure 2.3 on page 12). Lately, more and more algorithms have started using elitism, which allows for highly fit parents to survive and not be replaced by some less fit offspring. A selection scheme using $(\lambda + \mu)$ selection would select the parent population for the next generation based on the best individuals amongst the combined parent and offspring individuals (see the case with the short stroked lines in figure 2.3 on page 12). The elitism can, however, also be partially implemented such that a highly fit parent population can be prevented from excluding good offsprings from being selected. For these intermediate elitist selection schemes there does not yet exist a generalized notation. This is mostly due to the fact that the notations (λ, μ) and $(\lambda + \mu)$ are borrowed from evolution strategies which do not allow for intermediate elitist selection schemes due to their purely deterministic structure (see section 2.7.3 on page 35 for further details).

As mentioned previously, selection alone is not enough to evolve good solutions. The selection operator is limited by the contents of the initial population, and if given enough time the use of selection alone would result in the population converging on the best individual of the initial population. This is where the genetic operators come into the picture and they will be discussed in the following section.

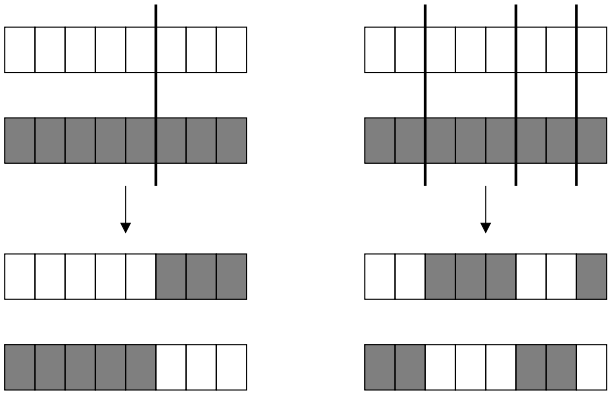
2.5.4 Crossover

Crossover is the genetic operator used when producing offspring by recombination of two parent individuals. As such, crossover can be considered as another name for the sexual genetic operator. This operator was inspired by the way most higher species reproduce in nature, and is thus considered as a crucial element to the evolutionary process. Exactly how this crossover, or recombination, is performed can, depending on the implementation, vary quite a bit.

For the binary case, the most common crossover method is the single point crossover, where a single point is randomly chosen in both parent individuals and the genetic material on one side of that point is swapped with the other parent, thus producing two offspring. This simple crossover can be expanded to the case of n-point crossover, where n points are chosen and the genetic material between every other crossover point is swapped. In figure 2.11 on the following page an example of single point and triple point crossover can be seen, but the principle can be extended to other cases where the number of points can vary from 1 to $l - 1$, with l being the length of the individual, either as a binary string or a number of indivisible elements contained in an individual.

Another type of crossover for the binary case is uniform crossover where each bit-position in the parent individuals are swapped with a certain probability, usually 0.5, but other probabilities can be used as well. This kind of crossover can thus be considered as a variable point crossover where probabilities for crossover near 0.5 would result in many multi-point crossovers and very high or low probabilities would result in more crossovers having a smaller number of crossover points. For this method, the elements that are swapped will generally be small segments. In figure 2.12 on the next page, the principle of uniform crossover is shown for a crossover probability of 0.25.

For the binary case, the crossover operator can be considered as a random shuffle (Goldberg, 2002). Also, if the crossover is extended to the panmictic case where more than 2 parents are used to create offspring, the crossover can be performed in a similar way to the methods described earlier and the result would still be a random shuffle of the genetic material in the parental popu-



(a) Single point crossover. (b) Triple point crossover.

Figure 2.11: Examples of single point and triple point crossover.

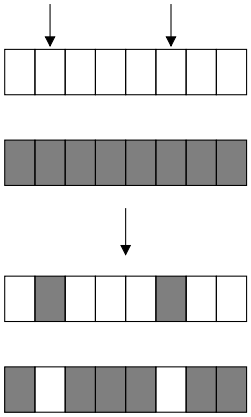


Figure 2.12: Uniform crossover with probability 0.25.

lation.

For real-valued genetic algorithms, the crossover cannot be done using genotypes, since the representation consists solely of phenotypes. This means that the crossover operators for the real-valued GAs are somewhat different than for the binary case. The real-valued equivalent to the binary n-point crossover is the naive crossover, which uses the same principle for the real-values of a phenotype as the binary case used for the individual bits (see figure 2.11 on the facing page). This method is, however, severely limited, since the number of real-values in the representation is much smaller than the equivalent binary representation would have for the same problem.

Other widely used real-valued crossover operators exist, such as blended crossover (BLX) and simulated binary crossover (SBX). For BLX, the idea is to find offspring individuals which have elements that are uniformly distributed in an area around or between the elements of the parent individuals. This can be seen in figure 2.13, where the BLX operator is illustrated for a single element of an individual and where the element of the offspring will be placed somewhere in the designated interval given by the thick line.

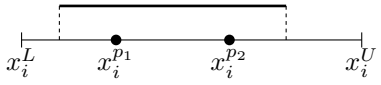


Figure 2.13: BLX crossover operator where the parental elements are marked with circles and the offspring elements will be located with uniform probability on the interval given by the thick line. The upper and lower bounds for the x_i elements are designated x_i^U and x_i^L respectively.

The exact location of the elements of the offspring is based on a uniformly distributed random number, which thus results in the offspring created by the BLX operator to be uniformly distributed in the area around or between the parent individuals.

On the contrary for SBX it is, as the name implies, attempted to simulate the effect of single point crossover on binary strings. For binary strings, the single point crossover produces offspring with phenotypes that are non-uniformly distributed near one of the parent individuals in the genotypic space. In order for an offspring to be located far from both of the parents, it is required that the parents are located far from each other. The position in which the crossover is performed will also have a large influence on whether or not the offspring will be located far from the parents or not. Only if the crossover is performed in a position which will ensure that the differences between the parents are mixed in the offspring will it be possible for the offspring to be located far from the parents. As such, for the binary case the offspring will have a higher probability to be located in the vicinity of one of the parents and it is this effect which SBX simulates. This is illustrated in figure 2.14 on the following page which indicates two different possibilities for the distribution of the elements of the offspring near the parents depending on the parameter settings of SBX.

For further explanation of SBX with corresponding formulas for determining the probability density functions the reader is referred to Deb (2001).

As it was possible for the binary case, it is also possible to extend BLX to the panmictic case. However, for SBX this would be a cumbersome task to do and has not yet been done for real values. In general, not many GAs use panmictic genetic operators and the reason for this might lie

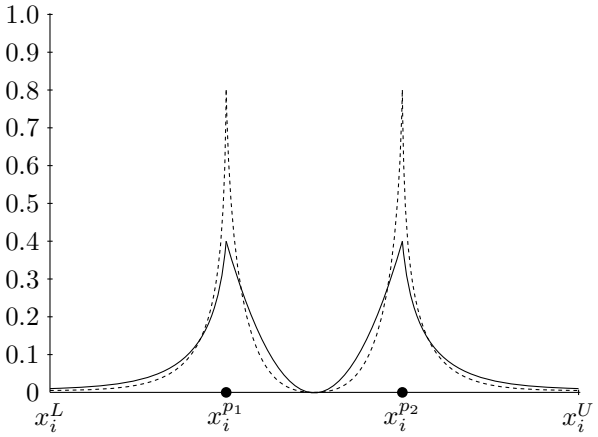


Figure 2.14: SBX crossover operator, where the parents are marked with circles and the placement of the offspring is given by the shown probability density functions, depending on what parameter settings is used.

in the fact that no evolutionary process so far has been known to be panmictic. Since the panmictic genetic operator has not proven itself useful in nature, the usefulness of the operator for evolution, both in nature and artificially, is thus questionable at best. The use of a panmictic operator might also lead to premature convergence in some cases and this would then count as yet another reason for why the panmictic operator is so rarely used.

In contrast to the rarely used panmictic genetic operator, the asexual operator, also known as mutation, is used extensively and this operator is what will be discussed next.

2.5.5 Mutation

Mutation is a more common name for the asexual genetic operator. When performing mutation a single parent individual is taken as input and altered/mutated such that an offspring is produced. In reality, the asexual reproduction of an individual would ideally be an exact copy of the parent, which is also the way nature attempts to perform this self-replication. However, due to external factors, such as radiation or the fact that making copies of copies cannot always be exact, a certain amount of the self-replication efforts will result in small changes to the offspring, which then will be a mutation when compared to the parent. Because mutation is considered to result from small changes, the probabilities for mutation in a GA are usually set quite low so that the majority of genetic material of the parent will still be present in the offspring, thus maintaining a high similarity. As in nature, the offspring produced by mutation will usually have properties very similar to the parent, but in some cases the mutation can also result in radical changes.

For the binary case, mutation is usually implemented as a bit flip as illustrated in figure 2.15 on the facing page.

The effect of mutation on a binary string can be considered as a random walk in genotypic space

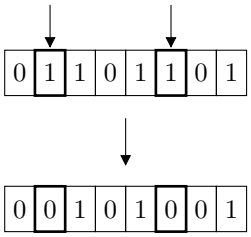


Figure 2.15: Mutation on binary strings using bit-flipping.

(Goldberg, 2002). One thing that makes the bit-flip approach of mutation significantly different from the equivalence in nature is the exponential structure of binary strings $(2^0, 2^1, 2^2, \ldots)$. As such, the mutations performed in a GA can in some cases have a major impact on the corresponding phenotype of the offspring, whereas nature usually does not introduce such drastic changes as often. This is mainly due to the fact that a bit-string in a GA will have a finite length that seldom exceeds 1000 bits, whereas for comparison a human genome consists of more than 3 billion bases (A, C, G, and T) (Human Genome Project, 2004). So, even if a mutation is more likely to occur in the human genome, it is very improbable that a single mutation will result in a drastic change, whereas the same cannot be said about the mutation in a GA.

For real valued GAs there are several different implementations of the mutation operator. One such implementation is the non-uniform mutation, which tries to mimic the effect of the bit-flip approach for the binary case. Each element of the offspring individual is biased towards the corresponding element of the parent and will thus have higher probability of being placed near that element of the parents as shown in figure 2.16.

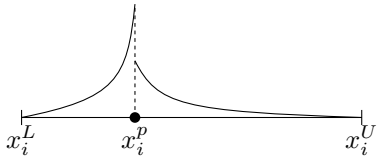


Figure 2.16: Probability density function for an element of an offspring using non-uniform mutation on a real valued individual.

Another type of mutation for the real valued case, inspired by evolutionary strategies (see section 2.7.4 on page 36), is performed by addition of a normally distributed random number with 0 mean and variance σ_i, $\mathcal{N}(0, \sigma_i)$, to each of the elements contained in the individual which will result in a distribution of elements as shown in figure 2.17 on the following page.

Other methods based on polynomials are also widely used and for further information on those the reader is referred to Deb (2001).

This concludes the basic description of a GA and its relation to the general EA. There are, however, a few additional issues tied into the workings of a GA that are quite relevant. These issues will be discussed next.

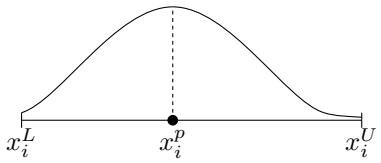


Figure 2.17: Probability density function for an element of an offspring using normally distributed mutation on a real valued individual.

2.5.6 Further Issues

Some of the areas in GAs are still undergoing research to continually improve different facets of the evolutionary process. A short discussion of those areas which are most relevant will hopefully help the reader to understand some of the underlying aspects to what makes a GA so successful.

2.5.6.1 Underlying Representation

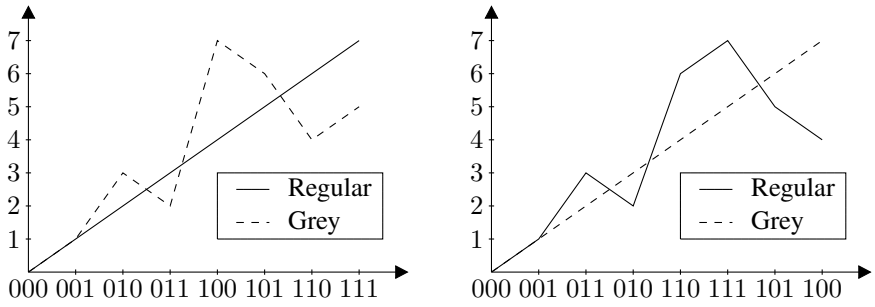
One area that has received quite a lot of attention is how the binary representation should be represented. Should the binary representation use a regular binary structure or should it for instance use Grey encoding. A comparison of regular binary encoding versus Grey encoding is shown in table 2.1.

Regular	Grey	Value
000	000	0
001	001	1
010	011	2
011	010	3
100	110	4
101	111	5
110	101	6
111	100	7

Table 2.1: Regular versus Grey encoding of binary strings

The main reason for this discussion is that for Grey code the Hamming distance, which counts the number of places where two binary strings differ in values, are always 1 whereas the Hamming distance for the regular binary case varies. However, as can be seen in figure 2.18 on the facing page, it is the structure of the problem, namely the way the evaluation function is implemented, that should decide which representation is used. The figure clearly shows that if the evaluation function is monotonically smooth with respect to regular binary encoding (figure 2.18a), then the corresponding problem using Grey code would become non-monotonic and thus more difficult to solve. Correspondingly, in a situation where the evaluation function is monotonically smooth with regard to Grey code (figure 2.18b) a binary encoding would make the problem more difficult to

solve.



(a) Smooth function for regular binary encoding.

(b) Smooth function for Grey encoding

Figure 2.18: Smoothness of functions depend on the underlying representation.

Another issue with GAs also relates to the representation. It is actually how the real valued GAs can exhibit similar behavior to GAs using binary representation. If there are intervals for the real values in which the GA performs better than average, these intervals can give rise to regions of better performance. These intervals can thus be considered as a virtual alphabet with lower cardinality, which in turn can be considered in a similar manner to the GAs based on binary representation (Deb, 2001). An example of how virtual characters can emerge in a two-dimensional space can be seen in figure 2.19.

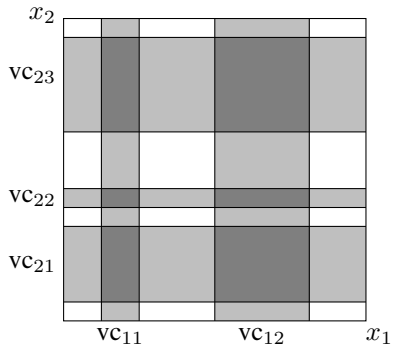


Figure 2.19: Virtual characters emerge as the intersection of intervals of real values which have higher than average fitness. This gives the real valued GAs a lower virtual cardinality which allows for theory based on binary representation to be applied to real valued GAs. Here shown for a two-variable space.

Besides the aforementioned representations and operators, there exists a group of GAs which

do not use specific genetic operators. These GAs are the so-called Probabilistic Model Building GAs (PMBGA). Instead of using crossover and mutation, these algorithms work by building probabilistic models over the parental population and creating the offspring population based on the generated probabilistic model. This has lead to one of the most powerful evolutionary based computation algorithms yet, namely the Bayesian Optimization Algorithm (BOA) proposed by Pelikan, Goldberg and Cantú-Paz (Pelikan, Goldberg, & Cantú-Paz, 1999) and the later improvements, such as the hierarchical BOA (hBOA) (Pelikan & Goldberg, 2001).

2.5.6.2 Building Blocks

In the field of GAs, Goldberg have long been an adamant advocate for the use of Building Blocks (BBs) (Goldberg, 2002). The basic idea is that the genome can be viewed as having several subgroups (BBs), whose performance depend solely on the internal contents of that particular BB. As such, the problem can be considered as a collection of subproblems which when each subproblem has been optimized, will result in an optimal solution for the full problem. This BB-wise approach have resulted in a comprehensive amount of theory and several different GAs. The theoretical issues include

- BB supply
- BB mixing
- Linkage

The issue of BB supply deals with population sizing and the way a sufficient supply of BBs can be assured so that good BBs will not be lost due to noise in the early stages of the GA. An extensive amount of work in this field has recently been done in Goldberg, Sastry, and Latoza (2001). For a generic GA, the population size is usually chosen medium-large depending on the problem difficulty and can thus vary from 30-50 in the smaller range to 1000 which is the upper limit usually encountered.

BB mixing should be optimized so that the good BBs can be found quickly and recombined with other good BBs early on, thus resulting in a fast evolution of good solutions. This also deals with the schema theorem first identified by Holland (Holland, 1975) and which have been widely used and promoted by Goldberg ever since. This has received a considerable amount of attention when trying to find the so-called "sweet spot" of a GA, which is an attempt to find the most appropriate relationship between the probability for crossover and the selection pressure of a GA.

The linkage problem has to do with the compactness of each BB. If the elements of a BB are spread widely across the entire genome, the BB will easily be disrupted by crossover operations, and the linkage for that case is said to be loose. If, on the other hand, the elements of a BB are tightly packed in a small area of the genome (the linkage is tight), it is less likely to be disrupted by crossover and will have a better chance of surviving throughout the evolutionary process. The only exception to this consideration exists for uniform crossover which will disrupt both tight and loose linkage BBs to same extent. One algorithm in particular which have been dealing with the issue of linkage is the Linkage Learning Genetic Algorithm (LLGA) proposed by Harik (Harik & Goldberg, 1996).

2.5.6.3 Steady State GA

Some algorithms use a special steady state GA structure, also known as a non-generational GA. The idea in these algorithms is that only one offspring is created at a time, and once the offspring has been created it is either inserted into the parent population, where it replaces the worst individual there, or is discarded. As such, the notion of generations do not apply to this case. It is, however, still possible to fit this kind of algorithm into the general GA structure, since the selection mechanism can be described as $(\mu + 1)$.

2.5.6.4 Multimodality

The subject of multimodality was first raised in Goldberg and Segrest (1987) and further discussed by Goldberg in his 1989 book (Goldberg, 1989). Suppose a function does not have one single optimum value but many. How is it then possible for an algorithm to converge on an optimum solution? First, take a look at figure 2.20.

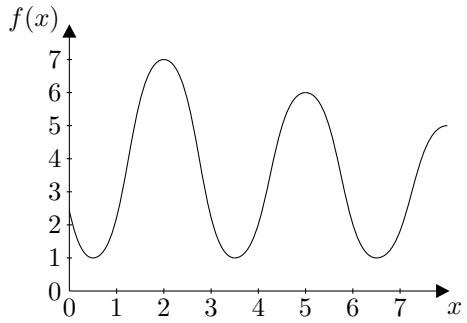


Figure 2.20: A function that is multimodal with respect to the variable x.

Suppose the function is sought to be maximized, then it is clear that the optimal value for x is 2 with a corresponding $f(x)$ of 7. However, what will be the optimum value if the function were to be minimized. For such a case there is no single best value for x and genetic drift would occur. In this case, without any clear best solution, the algorithm would drift around on the different optimal values and finally settle on one of the values, since there would be no selection pressure to guide it to one specific value.

To overcome the drifting problem with regard to the multimodality issue, Goldberg suggested the use of niching (Goldberg, 1989) such that identical values of $f(x)$ for values of x located far apart would be preferred to those values of x located near each other. This approach was successful in dealing with the multimodality issue. In section 2.10 on page 41 it will become apparent how the issue of multimodality also applies for algorithms dealing with multiple fitness functions and how the method of niching has been used and also inspired a similar strategy when dealing with those situations.

2.5.6.5 Learning Classifier Systems

Another field of great importance and related to GAs is the field of Learning Classifier Systems. The groundwork for this field was also laid down by Holland in his 1975 book (Holland, 1975). Since then it has evolved its own niche in the EA society and a lot of research is done in the field to improve performance and accuracy for the LCS algorithms.

The way an LCS works is by basically classifying input-output relations. Given a certain input, usually with multiple parameters, it is desired to know which output category it falls under. This requires that the LCS can be trained using some existing data, but once this is done it should be able to correctly classify the output categories for different input not present in the training set. As such, LCSs can be considered as algorithms which try to find the underlying structure of a problem given a training set. In many ways an LCS can actually be considered as an equivalence to a fuzzy logic system. An illustration of how classification is performed in an LCS can be seen in figure 2.21.

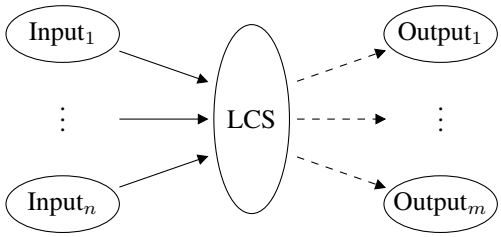


Figure 2.21: Illustration of a learning classifier system, where a combination of inputs results in one specific output.

In the figure, it is illustrated how a combination of input values will result in one specific output. The outputs are thus the classes into which the inputs must be classified. The performance of the LCS is then decided based on how well the LCS is at classifying the elements of the training set. The assignment of fitness to LCS is highly related to the area of reinforcement learning, where a good performance is rewarded and a bad performance is disregarded or punished.

The representation for the rules in LCSs are usually binary but lately a lot of research have gone into finding better and more optimal representations. This is, however, highly dependent on the underlying structure of the problem, just as the best representation for a GA was dependent on the structure of the problem.

In the field of LCS there exist two major groupings, a Michigan and a Pittsburgh approach.

The Michigan approach was initially developed at the University of Michigan and in this approach, the entire population consists of individual rules. Thus, each individual represents exactly one rule, and the aim is then to optimize the performance of the entire population.

In the Pittsburgh approach, initially developed at the University of Pittsburgh, each individual consists of a set of rules and the population is thus a collection of rule sets. As such, in this approach it is sought to find an optimal individual.

There are both strengths and weaknesses to each of the approaches. The Michigan approach

can easily get stuck in local optima, since it is the interaction of all the individual rules in the population that must yield an optimal performance. This weakness is avoided in the Pittsburgh approach, but at the cost of computational efficiency. Since the population consists of entire rule sets, it requires a lot of computation and as such it can be quite computational heavy.

When using LCSs, it is thus necessary to consider which of the approaches would be best suited to solve the given problem at hand instead of just picking one at random.

Now, after this introduction into the field of GAs, it is time to take a look at some of the other evolutionary based computational methods. As such, the time has come to give an introduction to an area closely related to GAs, namely genetic programming.

2.6 Genetic Programming

The basics of the field of Genetic Programming (GP) was already suggested by Holland in his 1975 book, but it was not until Koza embraced the subject in his book "Genetic Programming" (Koza, 1992) that it became widespread, even though some publications in the years prior had addressed the subject (Koza & Keane, 1990; Koza, 1991).

Just like the description for the GAs was split into a the underlying areas of representation, evaluation, selection, crossover, and mutation, so will this introduction to GP be split into the underlying areas starting with the representation.

2.6.1 Representation

The most significant difference of GP to other evolutionary computational methods is the representation which for GP consists of tree based structures. An example of this representation can be seen in figure 2.22.

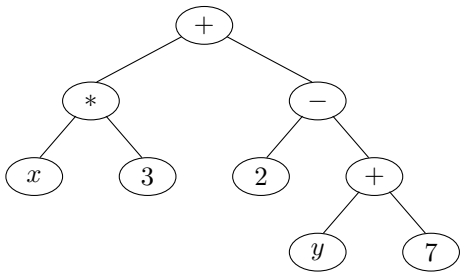


Figure 2.22: Representation using a tree based structure.

These trees consist of two types of elements, nodes and leafs. A node is a connecting element which connects the elements below using the function assigned to the node. A leaf on the other hand is an end point consisting of a value or input and has no further connections below. As such, the example shown in figure 2.22 contains 4 nodes and 5 leafs and the expression would read $(x * 3) + (2 - (y + 7))$. For a simple version of a GP tree with low cardinality, the nodes would

consist of the operators $+$, $-$, $*$ and $/$, where $/$ is a protected division operator which allows division by 0 by returning the value 1. For this simple version the input values contained in the leafs would normally consist of any real value or input variable. Even for such a simple case with low cardinality for the nodes the individuals are highly multimodal since a specific expression can be expressed in an infinite number of ways. Thus, there would be an infinite number of possible combinations for making up the elements of A_x.

Also, the tress are not limited to such simple cases. Not only can they be expressed as functions, which can be used in areas such as system modeling, but they can also be expressed as programs containing regular expressions and even automatically defined functions, loops, and similar. These are the reasons why the population sizes in GP usually are very large, in the range 10,000 to 100,000 or may even be bigger than that.

The cases presented by Koza are usually written as lisp expressions, since these allow for a simple yet convenient way when evaluating the trees and performing the different genetic operators on them. For the example in figure 2.22 on the page before the corresponding lisp expression would read as

$$\texttt{(+ (* } x \texttt{ 3) (- (2 (+ } y \texttt{ 7)))).}$$

Now with the representation of GP in place, let us take a look at the evaluation of these GP trees.

2.6.2 Evaluation

Evaluation of these genetic trees, Φ, depends on the context. If the context is to model a system and the tree thus represents an approximated model, then the evaluation is usually performed by minimizing a least squares error function over the range of possible input values.

For a classification problem where the trees represent a classifiers, the evaluation can be done by using raw fitness, which counts the number of correctly classified instances versus the instances that are incorrectly classified.

In the case a program is evaluated, the program, which the tree represents, is run through using the execution parameters and based on whether the performance of the program was good or bad a corresponding fitness value can be assigned.

As such, the evaluation of GP depends solely on the context in which the algorithm is used much identical to the way evaluation of GAs depend on the context. The similarities of GP to GAs do not end with the evaluation, but also carries into the selection operator as will become apparent next.

2.6.3 Selection

Since the selection operator s does not depend on the underlying representation, but only on the obtained fitness values, the selection operators usually used for GP are identical to those of GAs. As long as the requirements for the selection operators are met, such as only positive fitness values are allowed for the proportionate selection operators, then the selection can successfully be applied. For further details regarding the selection operators the reader is referred to section 2.5.3 on page 17.

For GP, it is thus time to take a look at the different genetic operators that can be applied to the genetic tree structure starting with crossover.

2.6.4 Crossover

Because of the unique tree structure, the genetic operators, ω_{Θ_i}, for GP are different from any previously described. The simplest way of performing crossover on a tree based structure is by using cut and splice as shown in figure 2.23.

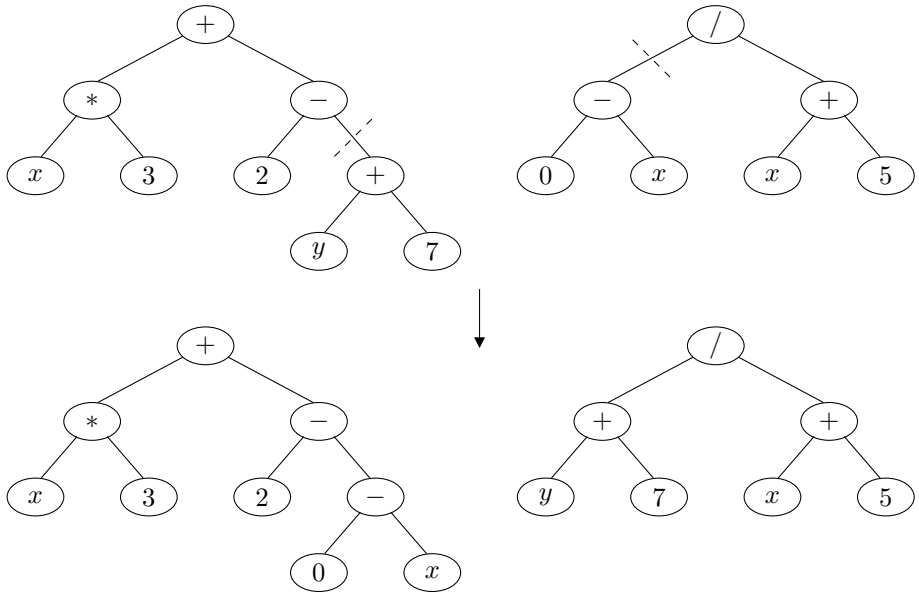


Figure 2.23: Crossover operator for genetic programming where the sections marked by the dashed lines are the subtrees that are swapped.

This method is quite effective, but for more complicated structures it can also make things quite complicated. If, for instance, the subtree that is used in the crossover is not suited to be placed at the desired place in the recipient, then how should the algorithm handle this. For this case there are two possibilities, either the input set for the nodes are extended such that they can accept any type of data which can be expressed in the tree, or the crossover operator is modified such that it can only allow crossing for subtrees of same data type.

Another vital issue with respect to crossover of subtrees in GP has to do with tree sizes. If there is not implemented a maximum for the obtainable tree depth, the trees can grow exponentially in size, which is usually not a good idea. When such a maximum tree depth is incorporated, a restriction must also be implemented for the crossover operator such that a crossover operation that would result in exceeding the allowed tree depth cannot be allowed to take place. This does,

however, limit the range of possible crossover operations for large subtrees, since that case would almost only allow crossover of subtrees of the same depth in the parental trees in order not to exceed the maximum tree depth.

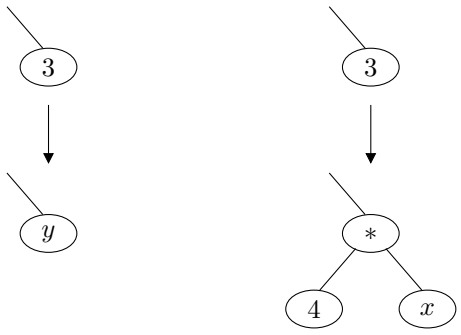
The crossover operator is, however, not the only operator for GP. The possibility for mutations also exist and will be described next.

2.6.5 Mutation

Even though most GP researchers only look at mutation as a background operator whose influence in the evolutionary process is limited, an introduction to the mutation operator will be given in the following.

For GP the mutation operator is quite different from mutation in GAs. As for real valued GAs there exist several different mutation operators in GP. They can, in principle, be divided into two categories, mutation of nodes and mutation of leafs.

When mutating leafs, they can either be mutated such that it remains a leaf and such that only the value of the leaf changes, which is usually called cycle mutation, or the leaf can be replaced by a node with a randomly generated subtree below, which is usually known as the grow mutation operator. The latter case can, however, only occur if the leaf is not located on the maximum allowable tree depth. In figure 2.24, it is illustrated how a leaf can mutate into either another leaf (figure 2.24a) or a node (figure 2.24b).

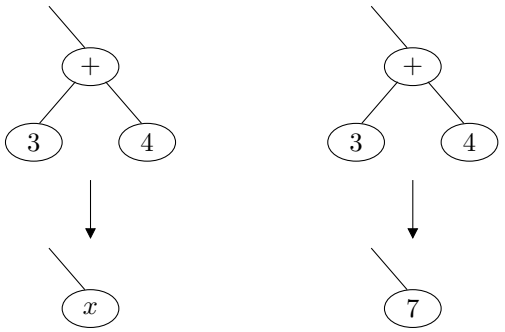


(a) Leaf to leaf mutation. (b) Leaf to node mutation.

Figure 2.24: Mutation of leafs into a node or a leaf.

For nodes, there are two corresponding options of mutating the nodes into either nodes or leafs. When mutating a node into a leaf, it can either be done by replacing the node with a random leaf or it can be replaced with a leaf whose value is set to the evaluated value of the subtree being removed. The latter case does not add diversity to the tree, but shortens the tree in such a way that the tree size can be kept below the maximum allowable size, thus opening up for

additional genetic operations without meeting any size restrictions on the tree. In figure 2.25, it is illustrated how a subtree structure can be mutated into a random leaf (figure 2.25a) or edited into the evaluated expression of the subtree (figure 2.25b). The latter case is, however, only possible when no variables are present in the subtree.



(a) Node to random leaf. (b) Editing of subtree.

Figure 2.25: Mutation of nodes into a leaf.

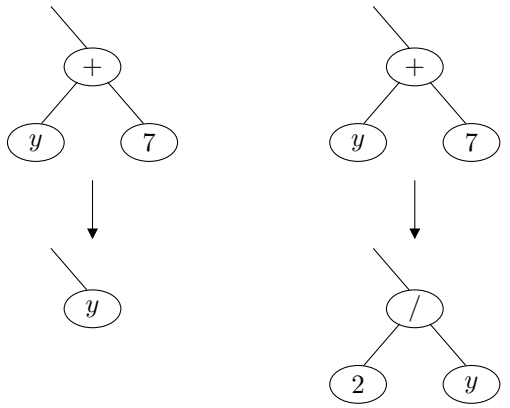
Mutation of a node into a node can be done either by replacing it with one of the underlying subtrees to that node, a truncation mutation, or it can be replaced with a randomly generated subtree, a replace mutation. These two cases are shown in figure 2.26 on the following page.

Figure 2.26a shows how a subtree, in this case the subtree is only a leaf, can be used to replace the original node, thus resulting in a truncation of the node. For the case of figure 2.26b the node is replaced by a randomly generated subtree. So even though the importance of mutation in GP by many is not considered important, there exist a large number of possibilities of performing the operation.

This description of the mutation operators in GP was the last of the subjects in this short introduction to GP. Hopefully, the reader now has a deeper understanding on how the fields of GAs and GP are alike and in which ways they differ from each other. It is now time to give a description of the remaining evolutionary computation methods, which will begin with a description of evolution strategies.

2.7 Evolution Strategies

The groundwork for Evolution Strategies (ESs) was laid by Bienert, Rechenberg and Schwefel in the 1960s at the Technical University of Berlin where the method was applied to hydrodynamic optimization problems that could not be solved analytically (Lichtfuss, 1965; Rechenberg, 1995; Schwefel, 1965). The development of ESs has since then been continually improved with regard to selection strategies and genetic operators, which will be discussed further in the next few sections.



(a) Truncation mutation. (b) Replace mutation.

Figure 2.26: Mutation of nodes into a node.

However, one of the major features of ESs has been the self-adaptation which, among other things, will be discussed in the next section. The self-adaptation is one of the major issues that still sets ESs aside as the continuing development of both ESs and GAs have resulted in algorithmic approaches which have become very similar.

Just like the way the introduction to the previously described evolutionary methods was performed, so will this description of ESs be split into subjects concerning the representation, evaluation, and operators. As such, the representation of ESs will be discussed next. It should be noted that many of the details in the following description of ESs are mainly gathered from Bäck (1996) and Deb (2001) and the figures are inspired by those given in Bäck (1996).

2.7.1 Representation

The representation of ESs has always been real values. Each value in an individual thus represents a parameter. However, the individuals do not only contain the problem specific parameters, A_x, but also the operator specific parameters, A_o. This is what is illustrated in figure 2.2 on page 11. With these operator specific parameters included in the individual, the algorithm is capable of adapting the parameters during evolution thus becoming self-adaptive. This is quite effective since the algorithm itself becomes capable of adjusting to changes in the fitness function such that the evolution of solutions can be kept near optimal. This self-adaptation is considered to be the strength of the ES approach.

However, as with the other evolutionary based computational approaches ESs must also assign fitness values to the individuals, and how this is done is discussed next.

2.7.2 Evaluation

The evaluation, Φ, of individuals in ESs is very similar to how evaluation is performed for GAs. The individuals are assigned a distinct fitness value depending on how well it performs in the context specified by the problem being solved. Once again, the use of fitness values would be pointless unless the values were used to distinguish good from bad individuals and as such the selection operator of ESs comes into the picture.

2.7.3 Selection

In the beginning ESs used a very simple two-membered structure. This means that there was only one parent that was mutated and the resulting offspring would then be compared to the parent in a selection very similar to the tournament selection of GAs (see section 2.5.3 on page 17). Thus, if the offspring was better than the parent then it would replace it and become the parent for the next generation. On the other hand, if the offspring was worse than the parent, then the offspring would be discarded and the old parent would again be used as parent for the next generation. This form of selection can be written as $(1 + 1)$ selection, and is thus an elitist strategy with one parent and one offspring. It is, however, worth to note that this kind of selection and those later developed in the field of ESs usually are purely deterministic. This fact is one of the major differences between ESs and GAs, since GAs can be allowed to use probabilistic as well as deterministic selection methods.

When ESs was further developed, a $(\mu+1)$ selection strategy was introduced. In this strategy there exists a population of μ individuals which can be used for mutation and recombination and which produce a single offspring. The offspring would then replace the worst individual of the parental population provided that the offspring was superior to that individual. This selection method was, however, not widely used, but it was a stepping stone for the development of the $(\mu + \lambda)$ and (μ, λ) selection strategies for ESs.

It is these $(\mu + \lambda)$ and (μ, λ) ESs that are so much similar to GAs that the two types of algorithms some times can be hard to distinguish from each other. This is especially the case for e.g. self-adapting GAs or ESs with non-deterministic selection. The shift from the two-membered ESs to the multimembered ESs also means that the population size usually used in ESs has grown from a low size of one to a more medium size of around 20 to 100.

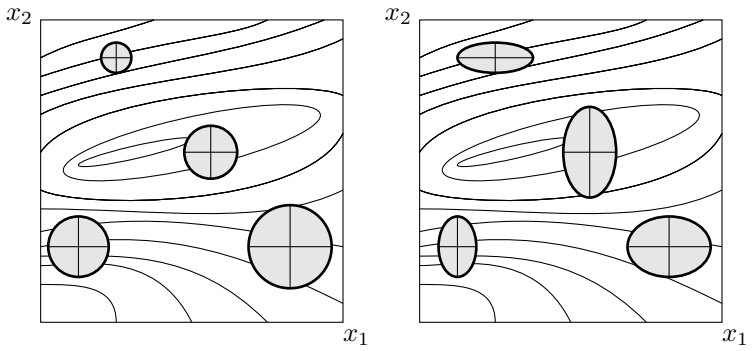
With regard to the advantages and disadvantages for the two different ES selection schemes $(\mu+\lambda)$ and (μ, λ) a rough comparison is performed in Bäck (1996) in which the conclusion is that (μ, λ) selection is superior to $(\mu + \lambda)$ selection. This has to do with the $(\mu + \lambda)$ selection having difficulties adapting quickly to changes both with regard to problem specific parameters as well as operator specific parameters, thus slowing the evolutionary process. It is also argued that $(\mu + \lambda)$ selection retains many local optima for an extended amount of time for multimodal functions, which also impedes evolution.

No matter which selection scheme is used, it cannot solve the problem alone and as such it is convenient to start discussing the genetic operators of ESs. The first operator to be discussed will be mutation since this was the only operator originally used in the two-membered ES.

2.7.4 Mutation

With regard to the genetic operators, ω_{Θ_i}, the mutation operator was the only operator used in the original ES formulation. In the very early ES formulation the mutation was only done with predefined step sizes, but it was soon changed into the version more commonly used today where the mutation is done according to a Gaussian distribution. As such, it was actually the mutation used in ESs that inspired the use of normally distributed mutation for real valued GAs (see section 2.5.5 on page 22). The main difference from the mutation used in ESs is though that the GA version of normally distributed mutation usually does not include operator specific knowledge. An example of the special mutation in ESs compared to that of GAs will thus be given in the following.

First of all, for each of the problem specific parameters there is usually a corresponding operator specific parameter specifying the variance of the normal distribution used when mutating that particular problem specific parameter. To illustrate why such a variance for each problem specific parameter is needed, take a look at figure 2.27a. The figure shows 4 individuals and their corresponding density functions of equal probability for mutations on the fitness function. The center of the circles indicate an individual and the shaded circles represent the area in which all of the mutations of that individual will occur. The curved lines in the figure are lines of equal fitness function value and can thus be considered as a height chart of the fitness function. Specifically, figure 2.27a shows the distribution of mutations for an individual with two problem specific parameters and only one operator specific variance. Now compare that with figure 2.27b in which the two problem specific values each have a corresponding operator specific variance, resulting in a deformation of the circular probability density areas into ellipsoids.



(a) Mutation using one σ value. (b) Mutation using two σ values.

Figure 2.27: Mutation distributions for individuals containing two problem specific parameters and having one σ value (a) or two σ values (b).

Figure 2.27 clearly shows that when each problem specific value have a corresponding σ value independent of the other problem specific values, it is possible to obtain mutations that are more

flexible and thus better able to adapt to the problem environment. This can be seen since the ellipsoids are better able to cover more lines of equal fitness without necessarily having to increase the area of the probability density functions with a factor of σ^2 which would be necessary for the circular case. This improvement is, however, not the only one possible.

There can also be also a number of correlation parameters, usually represented by rotation angles, which specify to what degree the different operator specific parameters influence each other when performing a mutation. This is illustrated in figure 2.28.

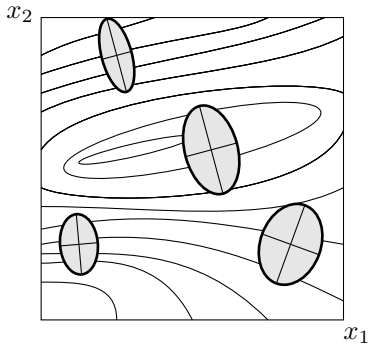


Figure 2.28: Mutation distributions for individuals containing two problem specific parameters and three operator specific parameters (two σ values and one correlation value).

It is clearly seen that a correlation value as part of the operator specific parameters allows for the mutation to adjust even better to the problem specified. In fact, the ellipsoids in figure 2.28 are rotated versions of those shown in figure 2.27b, and it is quite clear how much better the adaptation to the environment can be achieved when using correlation.

For this extended case, and if the correlation values are implemented as rotational values, an individual will contain the following parameters:

- n problem specific
- n operator specific variances belonging to $\mathbb{R}^+$
- $\frac{n \cdot (n-1)}{2}$ operator specific rotation angles in the interval $[-\pi, \pi]$

It is not a requirement that all of these parameters, used by the mutation operator, are implemented as part of the individual. For a case where some prior knowledge about the problem is known, e.g. if some parameters are independent of each other, this information can be used to reduce the number of correlation parameters and thus also improve the overall performance of the algorithm.

It should be noted that for ESs the mutation of operator specific parameters is done prior to mutation of problem specific variables. Whether this is significant with regard to the performance of ESs is an open issue, but the order in which mutation of the parameters is one of the significant elements which differs when comparing ESs with evolutionary programming.

With this substantial description of the mutation operator in ESs, it is time to take a look at the recombination operator as well.

2.7.5 Recombination

Recombination was not performed in the original two-membered ESs. As such, recombination was first introduced when the $(\mu + 1)$ ES was proposed. Of course, the use of recombination for $\mu = 1$ is not feasible and as such it was logical that the introduction of recombination for ESs did not occur until the time when $\mu > 1$ was used in ESs.

The multimembered ESs can both use sexual and panmictic crossover operators, but for simplicity the sexual crossover operator will be discussed first.

The simplest form of sexual crossover is the discrete recombination operator. This discrete recombination is similar to the crossover for the real-valued GA shuffling of elements, and also much similar to the uniform crossover with probability 0.5 performed in GAs using binary representation which can be illustrated using figure 2.12 on page 20. Thus, the elements from the parents in each position are shuffled, resulting in an offspring individual with elements from each of the parents present.

Another example of the sexual crossover operator in ESs is similar to the one previously described, but instead of shuffling the elements in each corresponding place the elements in the offspring will be the arithmetic mean of the corresponding elements of the parents. This is called intermediate recombination as is illustrated in figure 2.29.

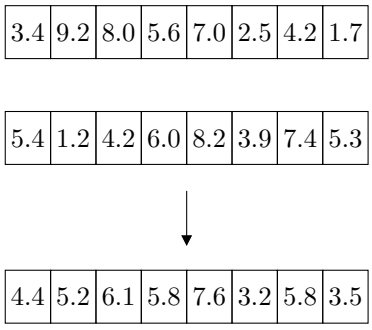


Figure 2.29: Illustration of intermediate crossover in a multimembered ES.

Correspondingly, a generalized version of the intermediate recombination operator was developed, which does not calculate the mean of the corresponding values ($x_i^o = 0.5 \cdot (x_i^{p_1} + x_i^{p_2})$), but finds an arbitrary value between the corresponding elements given by the formula

$$x_i^o = \alpha \cdot x_i^{p_1} + (1 - \alpha) \cdot x_i^{p_2} \quad , \tag{2.11}$$

where α is a uniformly distributed value in the interval $[0, 1]$.

Neither the intermediate crossover nor the generalized version of it have been implemented as panmictic crossover operators. However, when the multimembered ES was proposed, a panmictic

version of the discrete crossover operator was actually used when performing crossover. The panmictic version of the discrete crossover simply has a greater number of parents from which each element is shuffled into the resulting offspring.

This hereby concludes the description of ESs and the corresponding operators and it is thus time to give a brief introduction to the last of the evolutionary computation methods, namely evolutionary programming.

2.8 Evolutionary Programming

The last of the evolutionary computation methods to fall into the general EA framework is Evolutionary Programming (EP). EP was first proposed by L. J. Fogel in the early 1960s in Fogel (1962) and Fogel, Owens, and Walsh (1966). The representation used in those versions of EP consisted of finite state machines with discrete alphabets and selection was performed using a $(\mu+\mu)$ selection scheme. The only operator was mutation and the mutations were based on uniformly distributed random mutations on the underlying discrete alphabets. This original formulation of EP did not become very known at that time and was not used too extensively in a period of 30 years.

It was not until D. B. Fogel, the son of L. J. Fogel, took it upon himself to use his fathers methods and further developed them, before the field of EP became widely used. The extended versions which was developed in the late 1980s and early 1990s (Fogel, 1991; Fogel, 1992) did, however, evolve to be very much similar to ESs. This is not saying that they are a copy of ESs because the extended versions of EP evolved independently of ESs.

One of the major differences of EP when compared to ESs is that when mutation is performed in EP, the problem specific parameters are mutated before the operator specific parameters, whereas it is done in reverse in ESs. Also, the fitness function in EP is calculated quite differently. For each individual in the population a subset of size q from the population is chosen for comparison, and the fitness is then assigned according to how many of that subset which the individual was better than. A ranking is then performed according to fitness and the top half is selected for survival. As such, the fitness and thus also the way selection is ultimately performed becomes non-deterministic, whereas in ESs the evaluation and selection are mostly purely deterministic.

Finally, as the last difference between EP and the other evolutionary approaches described earlier, there is no recombination operator in EP. Whether the exclusion of the recombination operator is advantageous or not will depend entirely on the context and both arguments for and against this approach exists. The subject will, however, not be discussed further here.

Due to the high degree of similarities between EP and ESs, with the above exceptions, there will not be given an extensive introduction into this field. Thus, with EP being the last field of evolutionary computation in the original general evolutionary framework, it is now time to take a look at some of the more general extensions to the ordinary fields of evolutionary computation. One of the more special extensions to the ordinary field of evolutionary computation is the co-evolutionary algorithms, which will be discussed next.

2.9 Co-evolutionary Algorithms

The principle behind the co-evolutionary algorithms is how evolution of one species can be dependent on another. Examples of this can be found in abundance in nature. There are two distinct ways in which co-evolution comes into focus, namely through a symbiotic relationship or a predatory relationship. An example of a symbiotic relationship can be:

Example 2.7 (Natures cleaning crew) *In nature there are many examples where large animals, both herbivores and carnivores, get their skin or fur cleaned by allowing smaller animals to eat any parasites that might have become attached to the skin or fur. Thus, any small animals that have evolved themselves to be able to clean he skin or fur of specific animals can suddenly get a free source of food, whereas the larger animal reaps the benefit of avoiding irritation or even infections by removal of the parasites. The evolution of these animals are thus intertwined, since a change in one could greatly affect the other part, thus requiring the other part to adapt to the new situation.*

In the example it is clear, that even though the relationship is between two very different species, the continued evolution of either species will have a direct effect on the other. Similarly, the evolution of one of the species in a predatory relationship will influence the evolution of another. In this case, the evolutionary process can be considered as an arms race. This can also be illustrated using an example:

Example 2.8 (Arms race) *Let us consider the predatory relationship between an herbivore and a carnivore. In order for the carnivore to eat the herbivore it must first catch it. It is then clear, that the slowest of the herbivores are the ones most likely to be caught, since it would require less effort of the carnivores to catch them. This will thus shift the evolutionary process of the herbivores into creating offspring that are fast, since only the fast herbivores can survive long enough to procreate. This also means, that if the herbivores evolve to be faster for each generation then the evolutionary process of the carnivores will be affected, since the slow carnivores might be unable to catch any of the faster herbivores. Thus, the evolutionary process for the carnivores will bias towards becoming faster since the slowest of the carnivores will have trouble surviving because they cannot catch any of the faster herbivores.*

With the explanation above, it has become clear how evolution of different species can affect each other. There have also been quite a number of different publications using co-evolutionary algorithms for solving different problems (Hoffmann & Pfister, 1996; Ronge & Nordahl, 1996; Goldberg & Wang, 1997). The co-evolutionary approach is usually implemented as two separate populations evolving asynchronously as illustrated in figure 2.30 on the facing page.

When it comes to the evaluation of the individuals, a number of individuals from one population are used for computing the fitness values of the individuals in the other population, which is also shown as the dashed lines in figure 2.30 on the facing page. The populations thus interact with each other in order to solve the problem at hand. One example of an application which could take advantage of such a co-evolutionary approach is when designing controllers that need to be robust with regard to disturbances.

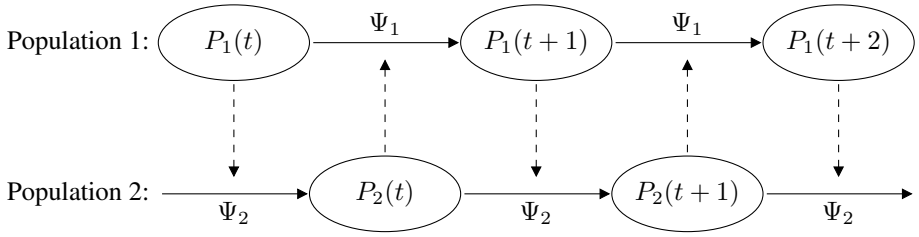


Figure 2.30: Overall structure of the co-evolutionary process where the dashed lines represent the influence of one population onto another.

Example 2.9 (Disturbance rejection) *For design of controllers that are capable of rejecting different disturbances, one population could consist of possible controllers for the system and the other population could be a collection of disturbances. This would bias the population of controllers towards those controllers who are best at rejecting the disturbances, whereas the population of disturbances would bias towards those disturbances which are best at upsetting the system. The result would then become a population of robust controllers as well as a collection of disturbances that are most likely to disrupt the system, thus giving additional insight into the possible weaknesses of the system.*

When it comes to the general EA framework as described in 2.4 on page 9, this co-evolutionary approach fits into the framework given, since co-evolution just consist of two populations running in parallel. The fact that the fitness functions for both populations can change during the evolutionary process does not exclude the co-evolutionary from the general framework, since there were no restrictions given on the collection of fitness functions Φ other than it should be a mapping from I to $\mathbb{R}^M$.

Besides from this co-evolutionary approach there are other extensions to the field of evolutionary algorithms. The extension discussed in the next section will give an introduction to the field of evolutionary algorithms using multiple fitness functions, also known as Multi-Objective Evolutionary Algorithms (MOEAs).

2.10 Multiple Fitness Functions

At this point it is prudent to emphasize on some of the terminology used in the remainder of this thesis, namely the difference between objectives and fitness functions in particular. Objectives can be considered as a representation of each different part of the problem to which a solution is sought. Fitness functions on the other hand are the actual implementation of both objectives and constraints into the algorithm. Thus, it is possible to combine one or more objectives or constraints into one or more fitness functions. Unfortunately, in order to comply with the terminology regularly used in the field of multi-objective optimization, when distinguishing between algorithms having only one or several fitness functions they will be referred to as single objective and multi-objective algorithms respectively. As such, when using the expressions single objective and multi-objective it will refer to the number of fitness functions used in the algorithm and

not the number of individual subproblems that is sought optimized. A single objective algorithm would thus have only a single element f_1 belonging to Φ with the mapping $\Phi : I \rightarrow \mathbb{R}$, whereas a multi-objective algorithm will have multiple elements $f_1, \ldots, f_M$ belonging to Φ with the corresponding mapping $\Phi : I \rightarrow \mathbb{R}^M$. However, when it comes to a discussion of the different parts of a problem they will be referred to as objectives or constraints depending on the relative importance. A more in-depth discussion of this is given in chapter 4 on page 73.

The different areas of evolutionary computation previously discussed have all been based on the case with only a single fitness function, the single objective algorithms. For some cases, this is enough and it is possible to settle for one of those approaches. However, the closer the EAs come to be applied to real world problems it becomes more and more apparent that it is necessary to consider the cases with multiple fitness functions as well. An illustrative example of where multiple fitness functions could easily have be applied was already given in example 2.1 on page 8, where it was explained how an herbivore needed both the ability to find food and the skill to avoid predators in order to survive. For those two separate skills (objectives) it would be natural to have two fitness functions each representing one of those skills. Relating this to the terminology, when using two separate fitness functions to represent the two skills (objectives) it would become a multi-objective problem, whereas if the skills had been combined into a single fitness function, representing a combination of those two skills, it would be a single objective problem.

In the field of control engineering, which is the field for which this research is aimed, almost all of the problems contain multiple objectives. By formulating these multiple objectives into multiple fitness functions, it becomes possible to obtain a set of optimal solutions taking all of the objectives into consideration rather than having to settle for a single solution. In many cases this approach can also help in getting a better understanding of the problem at hand. Interestingly, the only thing that is different for the case having multiple fitness functions compared to those having just one is the selection operator. However, before it can be fully explained how selection can be performed for the case with multiple fitness functions, it is first necessary to know how it can be determined which solution is better than another when there are more than just one fitness function. This leads to an introduction to the concept of dominance.

2.10.1 Dominance

When there is only one fitness function, it is possible to use the relations $<$, $>$ and $=$ to distinguish if one fitness value might be better than another or if they might be equal. As such, it is possible for the single fitness case to perform this comparison for all combinations of fitness values and based on this it can be determined which solution might be better than the rest. However, as soon as each solution is assigned more than one fitness value, it is no longer possible to use the simple one dimensional relations and it is thus necessary to introduce a new way of determining which fitness value is better than another. This is where dominance comes into the picture.

The following description of dominance conforms to the description given in Deb (2001), which is also the source for the definitions given in this section. First of all, it is assumed that the number of fitness functions is fixed at M. Then an operator $\lhd$ is introduced to indicate whether a solution is better than another. As such, the expression $x^1 \lhd x^2$ means that solution x^1 is better than solution x^2. This is done so that the same expression can be used regardless of whether it is a minimization or a maximization problem. Thus, for a minimization problem $x^1 \lhd x^2$ corresponds to $x^1 < x^2$

and for a maximization problem $x^1 \triangleleft x^2$ corresponds to $x^1 > x^2$. Similarly the expression $x^1 \triangleright x^2$ means that solution x^1 is worse than solution x^2.

It is then possible to define dominance as follows:

Definition 2.10.1 (Dominance) *A solution x^1 is said to dominate another solution x^2 if both conditions 1 and 2 are true:*

1. *The solution x^1 is no worse than x^2 in all fitness values, or $f_i(x^1) \not\triangleright f_i(x^2)$ for all $i = 1, 2, \ldots, M$.*
2. *The solution x^1 is strictly better than x^2 in at least one fitness value, or $f_{\bar{i}}(x^1) \triangleleft f_{\bar{i}}(x^2)$ for at least one $\bar{i} \in \{1, 2, \ldots, M\}$.*

For the case where x^1 dominates x^2 it can be written mathematically as $x^1 \preceq x^2$, but only if both conditions of definition 2.10.1 hold true.

In this connection, there are some properties that are worth mentioning concerning the dominance relation. First of all, it is not reflexive because it is not possible for an individual to dominate itself. Secondly, it is not symmetric since $p \preceq q$ cannot imply that $q \preceq p$, which means that if a solution p dominates a solution q then q cannot dominate p in return. In fact, the relation is asymmetric because the converse is true, namely $p \preceq q$ implies that q does not dominate p. Because the dominance relation is asymmetric, it follows that it cannot be antisymmetric, since it is not possible for $p \preceq q$ and $q \preceq p$ to imply that $p = q$. Further, it can be noted that the relation is transitive, meaning that given the fact that $p \preceq q$ and $q \preceq r$ then the relation $p \preceq r$ holds as well. Finally, it is worth noticing that $p \npreceq q$ does not necessarily imply that $q \preceq p$, understood such that if q does not dominate p, it cannot be concluded that p dominates q. The dominance relation is thus not total.

Similar to the dominance relation, it is possible to define a stronger version:

Definition 2.10.2 (Strong dominance) *A solution x^1 strongly dominates a solution x^2, if solution x^1 is strictly better than solution x^2 in all M fitness values.*

Mathematically, the strong version of dominance can be written as $x^1 \prec x^2$ if x^1 strongly dominates x^2. For the strong version of dominance, the same properties are valid as those that apply to the regular dominance relation.

Based on the above observations, both the regular and the strong dominance relation can be quantified as ordering relations, because the only requirement for this to be true is that the relations must be transitive. More specifically, both dominance relations are strict partial ordering relations since they are irreflexive as well as asymmetric. It then follows that the dominance relations do not qualify as partial orderings in the general sense, because they are neither reflexive nor antisymmetric but only transitive. These issues are quite important to keep in mind when the underlying strengths and weaknesses of the multi-objective approach will be discussed later in chapter 4 on page 73.

For simplicity, the concept of dominance for the two-dimensional case is shown visually in figure 2.31 on the following page, but in general the dominance relation is also valid for other dimensions as well.

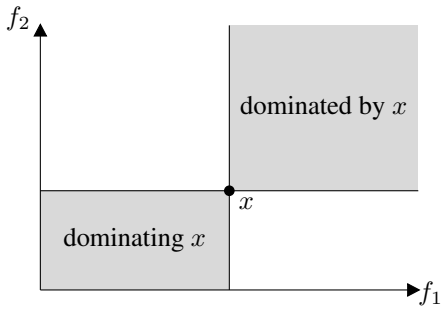


Figure 2.31: For the problem of minimizing the functions f_1 and f_2, the solution x is dominated by any solution belonging to the lower left shaded region, whereas x dominates any solution located in the shaded region to the upper right.

In the figure, where the fitness functions f_1 and f_2 are sought minimized, the solution x dominates any solution belonging to the upper right shaded region and x is dominated by any solution belonging to the shaded region to the lower left. Further, it can be seen from the figure that if the lines going through x are included as part of the shaded regions, with the exclusion of the point x, it corresponds to the first definition of dominance (definition 2.10.1 on the page before), whereas if the lines are not included in the shaded regions it corresponds to the definition of strong dominance (definition 2.10.2 on the preceding page).

In essence, the strong dominance relations $\prec$ and $\succ$ are element-wise extensions of $<$ and $>$ into higher order dimensions. However, a similar parallel cannot be drawn between the dominance relations $\preceq$ and $\succeq$ and the corresponding one-dimensional ordering relations $\leq$ and $\geq$ because of condition 2 in the definition of the dominance relation (definition 2.10.1 on the page before). This fact can also be realized when noticing that the ordering relations $\leq$ and $\geq$ are antisymmetric, whereas the dominance relation is not.

With the definition of dominance in place it is then possible to give an example of how the principle can be used on a population of solutions. The following figure (figure 2.32 on the facing page) shows five different solutions to a minimization problem.

In relation to the figure and because it is a minimization problem that is under consideration, it follows that the dominance relationship between the solutions can be set up as shown in table 2.2 on the facing page.

Elementarily put, it is possible in the table to see which solutions are better than others by noticing which solutions dominate some of the other solutions.

Based on the dominance relations given so far it is now possible to give the definition of a non-dominated set:

Definition 2.10.3 (Non-dominated set) *Among a set of solutions P, the non-dominated set of solutions P' ($P' \subset P$) are those that are not dominated by any member of the set P.*

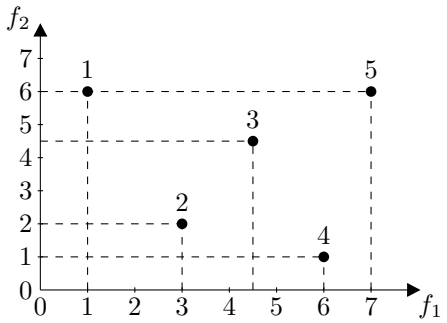


Figure 2.32: A two-objective minimization problem with five potential solutions.

Solution	Dominates	Strongly dominates
1	5	–
2	3, 5	3, 5
3	5	5
4	5	5
5	–	–

Table 2.2: Dominance relations for figure 2.32.

Applying this definition to the previous example it is possible to determine the non-dominated set to consist of solutions 1, 2, and 4.

Also, when relating the non-dominated set to the visualization of the dominance relations in figure 2.31 on the facing page it is clear that if solution x belongs to the non-dominated set, then any other solutions belonging to the non-dominated set must either be located in the non-shaded regions or be located in the same point as x. Thus, if any of the other solutions belong to the shaded region in the upper right, then they cannot belong to the non-dominated set, since they are dominated by x. Additionally, if any solutions are located in the lower left shaded region, then the assumption that x belongs to the non-dominated set will be incorrect, since there exist one or more solutions which dominate x.

Had the set P of definition 2.10.3 on the facing page been the entire feasible search space S then the resulting non-dominated set P' would be called the Pareto-optimal set. Thus, the following definition of the global Pareto-optimal set can be given:

Definition 2.10.4 (Globally Pareto-optimal set) *The non-dominated set of the entire feasible search space S is the globally Pareto-optimal set.*

For some problems it is also possible to obtain local Pareto-optimal sets similar to the way local and global extrema exist for functions. A local Pareto-optimal set can thus be achieved if it conforms to the following definition on the normed space of possible solutions:

Definition 2.10.5 (Locally Pareto-optimal set) *If for every member x in a set $\underline{P}$ there exist no solution y (in the neighborhood of x such that $\|y - x\|_\infty \leq \epsilon$, where ϵ is a small positive number) dominating any member of the set $\underline{P}$, then solutions belonging to the set $\underline{P}$ constitute a locally Pareto-optimal set.*

The definition given for the globally Pareto-optimal set thus also holds true for the definition of a locally Pareto-optimal, confirming that a globally Pareto-optimal set is also a locally Pareto-optimal set.

With the above definitions of dominance, non-dominance and Pareto-optimality the basis has now been laid for better understanding how the selection operator of different MOEAs can decide between the different individuals. Generally, it should be clear how decision making for the multiple fitness case reduces down to three possibilities no matter how many fitness functions there are. These possibilities are, x^1 dominates x^2, x^2 dominates x^1 or x^1 and x^2 are non-dominated. For the latter case, a tie-breaking procedure needs to be formulated, but for now let us take a look at a basic introduction to some of the selection operators used in different MOEAs.

2.10.2 Selection

For MOEAs there have been developed several different ways of performing selection. In this section a brief discussion of three of these selection methods will be presented. It should be emphasized that the description of the different selection methods will focus mainly on the overall principle of the methods rather than on details. For a full description of the measures which are used in these methods, it is encouraged to read the papers by the corresponding authors of the algorithms, where more details are available.

The first selection method to be discussed is the one used for the multi-objective algorithm given in Fonseca and Fleming (1993a). It describes the selection principle of one of the first algorithms to use dominance in combination with multiple objectives, namely the Multiple Objective Genetic Algorithm (MOGA).

2.10.2.1 Multiple Objective Genetic Algorithm

In MOGA, the rank of an individual i is given by $r_i = 1 + N_{d_i}$, where N_{d_i} is the number of individuals which dominate individual i. Thus, individuals which are non-dominated will have rank one, whereas an individual which is dominated by all other individuals will have the maximum obtainable rank equal to the population size μ.

In this case, an illustrative example would be quite helpful in understanding the ranking procedure. As such, it is shown in figure 2.33 on the facing page what result the application of the MOGA rank assignment procedure will have on a small sample population.

For each rank, a raw fitness based on a linear mapping is assigned to the individuals, such that the individuals with the highest rank gets the lowest raw fitness and vice versa. For individuals with equal rank the raw fitness values are averaged over these, such that the raw fitness for these individuals also becomes equal and this is called the assigned fitness.

Using this procedure it is clear that there often will be situations where more than one solution

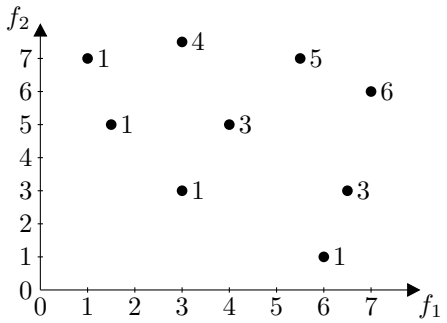


Figure 2.33: Example of Pareto ranking for MOGA.

will have the same assigned fitness. The problem has thus been reduced to a multimodal single fitness optimization problem and the way this issue is resolved in MOGA is by using one of the niching methods designed for dealing with multimodality in single objective algorithms.

The niching method used in MOGA for coping with the multimodality issue is the one known as niche count. The method is applied to those individuals, having equal assigned fitness values, in such a way that the individuals which reside in less crowded regions of the objective space are preferred to those located in very crowded regions. The niche count method uses a value σ_{share}, which is the extent of the niche around each individual. For those individuals, which lie within this distance σ_{share} from the current individual and have the same rank, the corresponding sharing values are summed and the sum then counts as the niche count of the current individual. The formula for calculating the sharing value is given in formula (2.12).

$$Sh(d) = \begin{cases} 1 - \left(\frac{d}{\sigma_{share}}\right)^{\alpha}, & \text{if } d \leq \sigma_{share} \\ 0, & \text{otherwise,} \end{cases} \tag{2.12}$$

where d is the normalized euclidean distance in the space spanned by the fitness functions between the current individual and the one which the sharing value is calculated for and $\alpha = 1$.

After finding the niche count, it is used in calculating the shared fitness which is simply the assigned fitness value divided by the niche count. This shared fitness is then scaled such that the average of the shared fitness for the individuals with the same rank is equal to the assigned fitness value.

After this comprehensive calculation of fitness values the result is a fitness value for each individual which is dependent on the rank and the distance between individuals with equal rank. Another important feature regarding this calculation of fitness values is that the sum of fitness values over all of the individuals in a generation is constant. This turns out to be advantageous because the selection scheme used for the actual selection is SUS and it is thus quite easy to apply this selection in the algorithm.

It should now be clear to the reader how the selection of MOGA was transformed from a problem having multiple fitness functions into a single fitness problem with multimodality. The major

difference from the single fitness case is, however, that the niching in MOGA was performed in fitness space whereas the niching for a regular single fitness problem can only be applied to either the phenotypic or genotypic space.

This was the first application of selection to MOEAs based on dominance. Another selection method which uses a somewhat similar dominance based ranking scheme can be found in the Non-dominated Sorting Genetic Algorithm II (NSGA-II) (Deb, Pratap, & Moitra, 2000) and will be discussed next.

2.10.2.2 Non-Dominated Sorting Genetic Algorithm II

In general, the selection mechanism in NSGA-II uses elitism and as such it uses a $(\mu+\lambda)$ selection scheme. Thus, once the genetic operators have been applied to a population $P(t)$ resulting in a population of offspring $P'(t)$ those two populations are combined into one pool from which only μ individuals will be chosen.

As a basis for performing the selection of this pool of individuals, a ranking procedure, which is an iterative version of the aforementioned principle of Pareto optimality, is applied.

At first, a check for dominance is performed on the entire pool of individuals. The individuals which belong to the non-dominated set are then labeled to belong to the non-dominated set, having a rank of 1. Then a check for dominance is performed on the remaining individuals of the pool which have not yet been labeled. The resulting non-dominated set resulting from that dominance check is then labeled to belong to the set with rank 2. This procedure is repeated until all individuals of the pool have been assigned a rank. As implied by the numbering of the sets, an individual belonging to a set with a low rank is considered to be better than an individual belonging to a set with a higher rank. The individual sets are also commonly known as fronts. An illustration of the fronts produced by the Pareto ranking procedure is shown in figure 2.34.

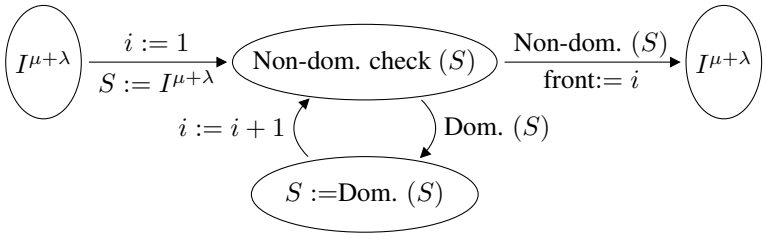


Figure 2.34: Procedure for performing Pareto ranking.

The application of the NSGA-II ranking procedure is performed on the same example as the one used in the description of the MOGA ranking procedure and the result is illustrated in figure 2.35 on the facing page.

When performing the selection it is thus possible to determine which individuals are better than others as long as they have different ranks. Selection is then done deterministically by copying the front with the lowest number to the next generation, provided that the number of individuals in the front is not above μ. This is then done for additional fronts until the next generation $P(t+1)$

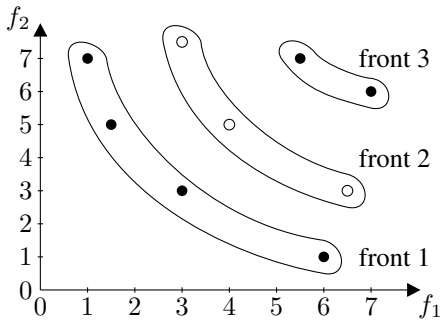


Figure 2.35: Example of ranking assignment using NSGA-II.

has been filled or until the inclusion of another front would result in exceeding the size μ of the next generation. This principle is illustrated in figure 2.36.

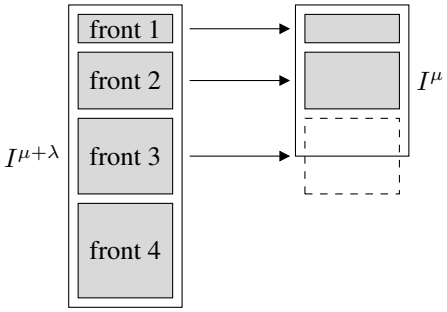


Figure 2.36: Filling up the next generation with preference for fronts with lowest rank.

The only remaining problem with this method is thus what should be done when the number of individuals in a front would make the next parent population larger than the allowed size of μ individuals. It is then necessary to select only the number of individuals from that front which will fill up the next generation, thus discarding other individuals from that same front. This once again leads to a problem similar to the multimodality issue for the single fitness case.

The way the drift is overcome in NSGA-II is by introducing a crowding distance, which is similar to the ideas of niching except that it does not require the parameter indicating the niche size. Thus, the application of crowding corresponds to a parameterless version of niching. The crowding distance d_j for an individual j is calculated as an estimate of the perimeter of a hypercube applied to neighboring individuals. The hypercubes are constructed for each individual j by finding the nearest neighboring individuals $j-1$ and $j+1$ in each direction for all fitness functions, and using these as vertexes for constructing a cube around the individual. This can then be used to estimate the perimeter of the hypercube, thus getting an indication of the size of the cube. In essence, the perimeter of the hypercube corresponds to the double of the Manhattan distance from individual $j-1$ to $j+1$. The bigger the perimeter of the hypercube is, the bigger the crowding

distance will be, and by preferring large values this will give preference to individuals which are located in non-crowded areas, since these will have the largest hypercubes. The construction of the hypercubes work by sorting the individuals in ascending order with regard to a fitness function i and finding the neighboring individuals for that fitness function. This procedure is then performed for all fitness functions. An illustration of the procedure is shown in figure 2.37 for a case having two fitness functions.

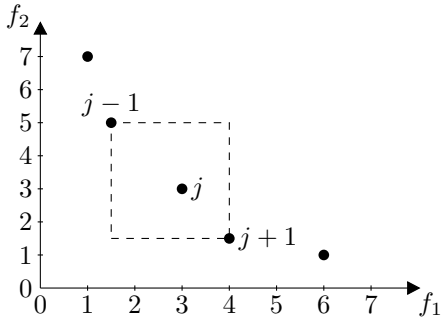


Figure 2.37: Illustration of hypercube for a two-dimensional case.

Formally, the crowding distance for individual j is calculated according to the formula in equation (2.13), and for reasons of efficiency the formula only calculates half of the perimeter size, corresponding to the Manhattan distance, since it will result in the same proportional relationship between the crowding measures as the one obtained using the full perimeter size.

$$d_j = d_j + \frac{f_{i,j+1} - f_{i,j-1}}{f_{i,max} - f_{i,min}} \quad , \tag{2.13}$$

where the crowding distance is summed over all fitness functions i. The denominator of formula (2.13) is a normalization term based on the maximum and minimum obtainable values for fitness function f_i. This normalization is included such that all fitness functions can influence the crowding distance even when the fitness functions have significantly different intervals in fitness space.

The individuals in an endpoint which do not have a neighbor to one side is assigned a value of ∞ such that the endpoints will always be preferred to inner points, thus preserving the full span of that particular front. With the crowding distances calculated, the individuals are then sorted according to those values and the individuals with the highest crowding values are selected for inclusion in the next generation such that the number of individuals reaches μ.

How the discussed aspects of the selection operator in NSGA-II interacts is illustrated in figure 2.38 on the facing page.

Provided that the evolutionary process has not ended, for instance by reaching the desired number of generations, there is still one more step for selection to perform, namely creation of the mating pool for $P(t+1)$, which corresponds to the operator ω_{Θ_0} of the general EA framework. This is necessary since the deterministic way in which the selection was performed, resulted in each of the survivors occurring only once in the new population. As such, the evolutionary process would

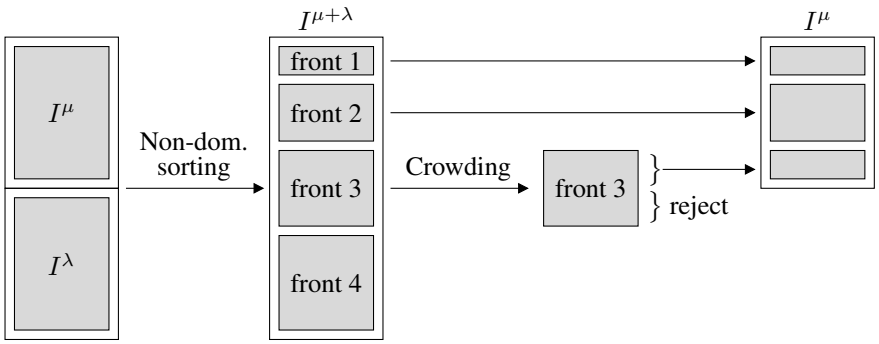


Figure 2.38: Illustration of selection process using NSGA-II.

become very slow because the best individuals of the new population would not be emphasized sufficiently over the worst individuals.

NSGA-II then uses a special tournament operator for this mating pool creation which actually consist of a two-step check. For a tournament with size two, the individuals are paired at random, such that all individuals have two occurrences among all the pairs and they are then pairwise compared to each other. If the individuals in a pair belongs to different fronts, the individual belonging to the lowest ranked front is chosen. In case the individuals belong to the same front, the one with the largest crowding distance is chosen. A similar procedure can be performed for tournament sizes larger than two where the individuals are compared in groups of three or more and each individual occur correspondingly many times in the union of all the groups. The default tournament size used in NSGA-II is fixed at two.

This concludes the description of the selection operator used in NSGA-II. It is now time to discuss a somewhat different method of keeping track of the best individuals which does not entirely fall under the category of selection, namely the use of external archives.

2.10.3 External Archives

This section will shortly describe a general concept used by quite a few MOEAs, namely the concept of external archives. This concept is used in algorithms such as the Strength Pareto Evolutionary Algorithm (SPEA) (Zitzler & Thiele, 1998) which was later improved to become SPEA-II (Zitzler, Laumanns, & Thiele, 2001), the Pareto-Archived Evolutionary Strategy (PAES) (Knowles & Corne, 1999) and also the Pareto Envelope-based Selection Algorithm (PESA) (Corne, Knowles, & Oates, 2000).

The idea is that there exists an external archive which is continually updated during the evolutionary process. If a new non-dominated solution is found, it is included in the external archive and solutions in the archive which might become dominated during evolution is deleted. As such, no evolution takes place based directly on the contents of the external archive, but it is used as an archive of known non-dominated solutions. To overcome the problem of a limited size of the external archive, the different algorithms use a crowding measure which can be based on cluster-

ing techniques or a hypercube grid. A detailed discussion of these crowding measures will not be given here, but the reader is encouraged to look in the corresponding papers describing the different algorithms.

For the evolutionary process, the external archive approach thus performs selection in a variety of ways, while in parallel a dominance and crowding based technique ensures that the non-dominated individuals found during evolution can be kept as well.

The external archive approach is thus another way of incorporation elitism, but without using the elitist individuals for further evolution. With this description of the use of external archives it is now time to take a look at how the multiple fitness aspect of MOEAs can be incorporated into the general framework, which was described earlier.

2.10.4 General Framework

In the general EA framework described in section 2.4 on page 9, a few loose ends were left with respect to the result of an EA or more precisely the operators Λ and Γ. Since it has now become known how the concept of dominance applies and how different MOEAs perform selection and preserves archives, it is now possible to give a fully detailed description of those two operators. During the following description it would be advantageous to keep figure 2.5 on page 13 in mind.

Let us first begin with the non-dominated operator Λ which is applied to each generation step. This operator is meant to describe the way individuals can be added to an external archive during the evolutionary process. As such, it corresponds to a non-dominated check for each generation and the non-dominated individuals are automatically selected for inclusion in the sets $A_0, \ldots, A_{\tau_{EA}}$.

The operator Λ can, however, not be considered alone. As mentioned in the previous section, the archive was updated with new solutions, but some of the old solutions were also deleted either because they became dominated or were located in crowded regions. This is where the preferential operator Γ enters the picture. The operator Γ in this case corresponds to yet another dominance check, but this time it incorporates a crowding mechanism, such that the resulting set does not consist of all the non-dominated solutions, but only those who also meet some crowding restrictions. As such, the combination of Λ and Γ allows for the resulting set to be an external archive as described in section 2.10.3 on the page before. Further, if the resulting set is continually updating during the evolutionary process then the contents of A_r will at all times correspond to the elements of the external archive.

For the algorithms which do not use external archives, like NSGA-II or MOGA, the non-dominated operator Λ is the same whereas Γ is modified. Since there is no external archive, the solutions of the individual generations can only survive if they continually survive in the active population and thus survives generation after generation. As such, the preferential operator Γ for these cases simply discards all the sets $A_0, \ldots, A_{\tau_{EA}-1}$ and the resulting set will thus be equal to the set $A_{\tau_{EA}}$.

With this description, the general EA framework has been extended to also cover MOEAs. This concludes the description of the general EA framework and how the different aspects of GAs, GP, ESs, EP and MOEAs are tied into this framework.

2.11 Summary

This chapter has been concerned with the basics of Evolutionary Computation (EC). First, a basis was established for why evolutionary computation is interesting when it comes to solving different problems. One of the main aspects in this discussion was actually that nature has had an immense success in developing the life on Earth as it is known today and as such proving how evolution can be a successful tool when it comes to development and problem solving.

Later, a general Evolutionary Algorithm (EA) framework was introduced. This framework was intended as a tool to allow the reader to realize how the different areas of EC are related and thus better understand the common structure of these different areas. One of the major improvements to the existing framework developed by Bäck (1996) was the inclusion of the multiple fitness algorithms.

It was then described how the different areas of Genetic Algorithms (GAs), Genetic Programming (GP), Evolution Strategies (ESs), Evolutionary Programming (EP) and Multi-Objective EAs (MOEAs) fit into the framework. During this description an in-depth view into those areas was given, allowing the reader to get a full understanding of the strengths, weaknesses, differences and further issues that are part of those different EC areas.

With this foundation, it is thus possible for the reader to get a better understanding of why EC is such a promising tool when it comes to solving control engineering problems. As such, the following chapter will give an in-depth introduction to areas within the field of control engineering which have already been subjected to different aspects of evolutionary computation.

Chapter 3

Evolutionary Computation in Control

Based on the comprehensive introduction to Evolutionary Computation (EC) in the previous chapter, it is possible to continue with a description of how EC over the years has been applied to the field of control engineering. Since this chapter is based on the introduction from the previous chapter, there will not be presented any new information with regard to EC. Only the different areas where EC have been applied to the field of control engineering will be presented. As such, it is possible to skip this chapter when first reading this thesis, since the information presented here will not be required in order to understand the remaining chapters. The chapter has been included because it will provide a perspective to the use of EC within the field of control engineering for which the research presented in this thesis is ultimately aimed at.

It may come as a surprise to many control engineers exactly how widespread the application of EC has been over the last couple of decades, but the fact is that EC has been used for control purposes in a variety fields. One of the first investigations of using GAs in the field of control was actually done by Hollstein (1971) in his doctoral dissertation from University of Michigan, where he referred to the method as genetic adaptation. A lot of the early work in using GAs and LCSs for adaptive control purposes was done by Goldberg in many of his late 1980s papers. This includes the optimization of natural gas pipelines done in Goldberg (1985a), Goldberg (1985c), and later in Goldberg (1987). He also tried using an LCS coupled with a GA for inertial object control and gas pipeline control in Goldberg (1985b). Furthermore, in Goldberg (1988) the scope of using GAs for different adaptive control purposes was investigated and showed a vast number of possibilities within the field.

This chapter will start out with a small overview of approaches using EC with a single fitness function for optimizing different control strategies such as optimal control, robust control, sliding mode control and related areas. This will be followed by a description of single-fitness EC used for different application areas such as control of autonomous systems, filter design, and topology design. After that, some of the hybrid approaches using fuzzy systems and neural networks will be discussed and following that the application of some methods related to EC within the field of

control engineering will be presented. The chapter will conclude with a description of some of the multiple fitness approaches that has been used for solving different control related problems.

3.1 Control Strategies and Evolutionary Computation

When it comes to optimization of existing control strategies using EC, it is usually the case that the controller structure has been fixed and it remains for the EC approach to optimize the controller parameters. Furthermore, the EC approach is usually used for offline computation of the parameters, since many processes are too fast for using EC optimization online. One reason, which causes many to be skeptical about using EC for online optimization, is the element of uncertainty which is inherent in EC. There is no guarantee to find the optimal solution, and since EC also needs to evaluate bad solutions this could result in major damages to a system. It is, however, relevant to see how EC has been applied to the different areas within control and the first area to be discussed here is the field of optimal control.

3.1.1 Optimal Control

Optimal control is one of the areas which have received the most attention when it comes to EC based approaches. As such, the description of the EC based approaches for this purpose will be divided into categories describing how the different EC areas have been applied to that problem.

Genetic Algorithms Some of the earliest attempts that used EC for parameter tuning of an optimal controller was done in Bailey and KrishnaKumar (1987) and KrishnaKumar (1988). Here, a GA was used for determining the optimum controller feedback gains for an energy based feedback controller for airplane control. In Bailey and KrishnaKumar (1987), the GA was applied to 12 different performance indices, which were all optimized individually, thus finding an optimal controller for different scenarios and variables. For KrishnaKumar (1988) the focus was more on energy concepts applied to control of an airplane flight in wind shear where the GA was used to find the optimum controller feedback gains for different performance indices prior to running flight simulations. This work was later extended in KrishnaKumar and Goldberg (1990) and also partly in KrishnaKumar and Goldberg (1992) where several aerospace control system optimization problems were solved using either a simple GA or a μGA (Micro-GA).

Other early approaches of using EC in optimal control include an attempt by Michalewicz, Krawczyk, Kazemi, and Janikow (1990) to use a real valued GA for finding controllers for a linear-quadratic problem and a harvest problem. The work was later extended in Michalewicz, Janikow, and Krawczyk (1992) where a push-cart problem was added. Also, Hunt (1992a) used a GA for finding optimized parameters for a simple optimal control problem, which resulted in a controller with performance very near the known optimal solution, and in Hunt (1992b) one of the issues was to use a GA for finding an optimal LQG controller. In another approach by Bobbin and Yao (1997), the task of finding an optimal controller was extended by adding switching costs so that chattering issues could be avoided in the evolved controller. Yet another way of finding the optimal control strategy was attempted in Smith (1995) and continued in Smith and Stonier (1996), where a GA was designed to optimize the input signals for a controller over a number of

time-intervals.

Learning Classifier Systems Another example of pioneering work was performed in Fogarty and Huang (1991) which used classifier systems for control purposes. One remarkable thing about this paper is that it used a Michigan style classifier for control of combustion in a simulated multiple burner installation. However, in the paper it was also tried to use a Pittsburgh style classifier to solve the cart-pole balancing problem. Later on Gilbert, Bell, and Valenzuela (1995) used a classifier system to optimize profit for a batch chemical process based on differing market conditions.

Genetic Programming A promising new step in the attempts to use EC for optimal control was performed in Koza, Yu, Keane, and Mydlowec (2000). In this paper, a GP was used to evolve a controller structure with a free variable for a three-lag plant where the free variable represented the time constant of the plant. The result was surprisingly good and the evolved controller had the topology of a PID-D2 controller followed by a first order lag. The controller even employed the technique of setpoint weighting where the reference was weighted before the plant output was subtracted from it. The evolved controller clearly outperformed a previously known best controller from Åström and Hägglund (1995). Similarly in Koza, Keane, Yu, Bennett III, and Mydlowec (2000), GP was used to evolve controllers for two-lag and three-lag plants with fixed time constants with good results.

Other Approaches Other approaches include the use of EP in Lai and Ma (1995) to find a unified power flow controller for optimal power flow control of a flexible alternating current transmission system. Further, in Chiou and Wang (1998) a hybrid version of differential evolution (Storn, 1996; Storn & Price, 1996) was formulated to find the optimal set of parameters for a bioprocess system.

Lately, several attempts have been made to use EC for optimal control of a green house climate. Examples of this can be seen in Pohlheim and Heißner (1999), Krink, Ursem, and Filipic (2001), and Ursem (2003).

So far, it has only been discussed how EC has been used for optimal control. However, by adding time as an element to be optimized, the problem formulation change and so does the potential solutions. How EC has been applied to time optimal control problems will thus be the focus next.

3.1.2 Time Optimal Control

When adding time as a constraint, the main problem usually becomes one of finding an optimal structure in which to perform the control. Since many time optimal problems thus become structural problems, the EC approach most often used to cope with the issue is GP. There are, however, always exceptions and one of these can be found in Curtis (1991). Here a GA was used in an attempt to find the time optimal bang-bang controller for a flexible system. One of the more conventional approaches was done in Howley (1996) where GP was used to find near minimum time control laws for two classes of spacecraft attitude maneuvers, namely rest-to-rest and rate-limited non-zero terminal velocity maneuvers. Further approaches using GP include Howley (1997),

which used it to evolve a time optimal controller for a two-link manipulator that was robust with regard to parametric sensitivity, and also Koza, Bennett III, Keane, and Andre (1997) which used GP to evolve an analog control circuit for the time optimal fly-to problem under constraints given by a maximal turn angle of an aircraft.

As indicated in the application of EC to the two-link manipulator problem, there is also a need for robustness with regard to uncertainties (Howley, 1997). It is then natural to continue with this issue next.

3.1.3 Robust Control Design

For the purpose of designing a robust controller, it is desired for the controller to be insensitive with regard to any disturbances which might occur either due to noise, parameter uncertainties, or uncertainties caused by modeling errors. However, for designing such a robust controller there exists no analytical way of calculating this which has given rise to the use of an iterative approach for such a design purpose.

Now, because many design procedures for robust controllers include such an iterative approach, it has been quite natural to use EC in the design process of these controllers. The application of EC to robust controller design has thus included many variations. In Saravanan (1995), an approach based on EP was used to find different H_∞ based controllers, whereas Ronge and Nordahl (1996) proposed to use co-evolution for development of robust controllers based on GP. GP was also used in Shimooka and Fujimoto (2000) to find robust control equations for the rolling inverted pendulum.

The use of GAs for the purpose of robust controller design has generally been the approach most people have tried to use. As such, using a GA for synthesizing a controller for the H_∞ mixed sensitivity problem was one of the issues in Hunt (1992b). Later on, Chen and Cheng (1998) proposed a simple GA which was used to synthesize optimal H_∞ controllers for two different systems, a phase-locked-loop motor speed control system and a longitudinal control system for a F18/HARV fighter aircraft. Keeping with the airborne machines, Dai and Mao (2002) used a GA to find an H_∞ controller for a helicopter such that it met the stringent level 1 handling requirements set forth in the ADS-33 aeronautical design standard. Also, in Schoen, Chinvorarat, and Schoen (2003), the attempt to find an optimal feedback gain matrix for robust control of a flexible space structure used a GA to solve the problem. In a combined fuzzy based GA approach, an H_∞ controller was sought synthesized based on a weighting of several different sensitivity functions in Donha, Desanj, and Katebi (1997). This last example is just one among many which all have used a hybrid combination of a GA and another method for solving different problems and more of such approaches will be discussed later. The next issue to be discussed is, however, more related to this section regarding control strategies, namely the field of sliding mode control also known as variable structure control.

3.1.4 Sliding Mode Control

The reason why sliding mode control is also known as variable structure control is because it switches between different controller structures based on the location of the current state of the system in the overall state space. This makes it possible to have the system slide along a trajectory

in the state space which is known to be stable and thus maintain stability.

The approach taken in Moin, Zinober, and Harley (1995) was to use the MATLAB GA toolbox to derive a static gain matrix for a sliding mode controller. Specifically, the evaluation of the different gain matrices was performed by looking at the maximum eigenvalues of an expression which ensured that the gain matrices actually complied with the sliding mode control. Another attempt to use EC for sliding mode control was performed in Kim, Kim, and Choi (1996). Here, an ES was used for finding a sliding mode controller for a brushless DC motor.

Sliding mode control as a whole can actually be viewed as an offline version of adaptive control. Where sliding mode control switches between a finite number of predetermined structures in order to adapt to the environment, adaptive control is used online to adapt the parameters to the environment and does not necessarily switch between different structures. The discussion of EC in control within the field of adaptive control is what will be discussed next.

3.1.5 Adaptive Control

Adaptive control is one of the only areas in which EC has been applied for online use. Because adaptive control is concerned with adaptation of the controller parameters online, it is essential for any EC based method to be faster than the dynamics of the system. For static systems this would not be too much of a problem, but for these cases it would often be a waste to implement an online adaptive control scheme unless it was to accommodate for fluctuations in parameters or for fault tolerant control. Since the EC used for adaptation must be faster than the dynamics of the system, it is important that the algorithm is fast or that the system is slow. As such, the main fields of EC which have been used online for adaptive control purposes are GAs and ESs.

In Porter II and Passino (1994), a GA was used to continuously calculate a control signal for a simulated cargo ship. Due to the slow dynamics of such a ship, it was a good example of a situation where use of EC online was possible. The control signal for the ship was calculated using a reference model and a plant model for the ship. In an attempt to ensure that good controllers are not lost, the algorithm was implemented with elitism. Another example of a GA used for adaptive control can be found in Cho and Gweon (1999). There, a GA was used for online optimization of the audio tuning process of a VCR. To accommodate for the uncertainties of the GA, the concept of age was introduced to act as a form of elitism. From these approaches, it also becomes clear that the issue of elitism has been important to ensure that once a good controller scheme has been found, it could be kept for several generations such that the performance level would not suffer from any sudden drawbacks caused by either drift or divergence of the population.

An example of using ESs for adaptive control was given in Park and Choi (1996) where it was used to implement an online PID controller for an unknown nonlinear dynamic system. The ES was used to identify the plant using an Auto Regressive Moving Average (ARMA) model for each sampling of the system as well as finding the optimal PID controller gains. The method was tested on a 200W DC motor with good results.

So far, the examples discussed have only considered controllers without regard to the order of the system or the controller. In the next section, a few examples will be presented which have used EC to synthesize reduced order controllers.

3.1.6 Reduced Order Controller Synthesis

There are several advantages of using reduced order controllers for control of higher order systems. First of all, if a reduced order controller can perform as well on a higher order system as a high order controller, then the dynamics of the system can be approximated with a lower order system which is easier to understand. With this simpler controller, it thus also becomes easier to recognize potential problems.

One attempt to use EC for synthesizing reduced order controllers for high order plants was performed in Caponetto, Fortuna, Muscato, and Xibilia (1994). Here, a GA was used in conjunction with a small gain theorem to find near optimal reduced order controllers of a fixed rank for high order plants. Another approach to finding reduced order controllers was given in Li, Tan, and Gong (1997). There, a GA combined with local optimization techniques was used for finding reduced order controllers. In this latter paper, it was attempted to find controllers of both fixed and non-fixed ranks for both fixed high order and infinite order systems.

Finding reduced order controllers is, however, not the only way EC can help with making controller design easier. Another way is to apply EC to stability issues, which is the subject of the next section.

3.1.7 Stability

The stability issue is one of the cornerstones of control engineering. When designing a controller, it is desired to be able to control the system and for this to happen, it is necessary to be able to stabilize the system. If a system cannot be stabilized, then inherently it is uncontrollable and there will thus not exist a suitable controller. As such, using stability as one of the elements when designing controllers is usually beneficial. Next, a few examples will be given of EC approaches which have been concerned with stability issues.

There are several ways in which stability has been applied to controller design using EC. In Tanaka and Hatanaka (1995), the stability was included as part of the fitness evaluation of a GA such that unstable controllers were not simulated, thus improving performance of the GA. Another way of using stability issues was investigated in Marra, Boling, and Walcott (1996). There, a stability analysis was performed on genetically evolved controllers, and using the schema-theorem (Goldberg, 2002) it was shown how the GA was expected to converge to a population consisting only of stable controllers under fitness-proportionate selection. The paper even gave an estimate on the number of generations needed for the population to have converged on only stable controllers. A totally different approach was taken in Lay (1994). There, a simple version of GP was used to search for multiple steady states for dynamical systems.

It should now be apparent how widespread the use of EC has been for optimizing different control strategies and other issues related to controller design. Some of the papers mentioned so far used EC for specific applications. In the next section, an emphasis will be put on the variety of applications which have benefited from the use of EC.

3.2 Evolutionary Computation in Control Applications

The list of applications for which EC has been used in one form or another is quite large and still growing. Some of the applications which will be discussed further in the following, include robot control, control of autonomous systems, filter design, system identification, scheduling and topology synthesis.

3.2.1 Robot Control

This section will describe some of the approaches in which EC have been used to evolve controllers for robots. First, a couple of examples will be given for control of non-autonomous robots, which will be followed by some examples where EC have been used to evolve controllers for semi-autonomous systems and also fully autonomous robots. After that, the issues of topology design, control of subsystems and evolvable hardware will also be addressed. Finally, the application of EC in the field of robot control will be concluded with some examples of EC used for control of multiple agents and design of test scenarios.

Non-Autonomous Systems The non-autonomous systems are those which are operated by a human operator, but where the underlying system is controlled by a controller which assures that an optimum level of performance can be maintained. In Alander (1991), it was attempted to find the optimal parameters of a GA such that it could be used for solving robot control problems such as path planing, etc. Later on, the same author gave a short review on the possibilities of using GAs for solving different robot control problems in Alander (1993). Since then, there have been several papers regarding the subject. This can be seen in Dakev, Chipperfield, and Fleming (1996) where a GA was applied to find the time-optimal path following control of a multi-body system. Also, both Kwok and Sheng (1994) and Porter and Allaoui (1995) used GAs for finding the optimal parameters for PID controllers of robotic manipulators. Finally, another approach to using EC for robot control was attempted in Gill and Zomaya (1995) where a simple GA was used for offline calculation of correction signals to compensate for dynamic variations between a model and an actual system consisting of a robotic manipulator.

Semi-Autonomous Systems There are also examples where EC has been used to find control rules for semi-autonomous systems, which are systems with a certain degree of automation but not fully automated. This include Grefenstette (1989b) and Grefenstette (1989a) where a classifier system was used to evolve control rules for a simulated plane such that it could evade an incoming missile. Another example is Dracopoulos (1997) in which GP was used to find a detumbling controller for a rigid body satellite. However, also an area such as 3D character animation has benefited from the use of EC. In Gritz and Hahn (1997), the task of animating 3D characters was achieved by considering the characters as robots and evolving controllers for these.

As mentioned before, the use of EC in robotics is not limited to control of non-autonomous and semi-autonomous systems. This will become apparent in the following where several examples of EC-based approaches to control of autonomous robots/systems will be discussed.

Autonomous Systems Many applications of EC within the field of autonomous systems have used the evolutionary approach to evolve behavior for different autonomous robots. An early example is Almássy and Verschure (1992) which used a GA for both on- and off-line path planning of an autonomous robot. Other papers using a GA for learning autonomous robot behavior include Ram, Arkin, Boone, and Pearce (1994), Hornby, Takamura, Hanagata, Fujita, and Pollack (2000), French and Damper (2001), and Uchibe, Yanase, and Asada (2001). Differently, a distributed classifier system approach was used in Dorigo (1995) for learning to control an autonomous robot 'AutonoMouse', whereas Shim and Kim (1995) used EP to find the optimal controller parameters for the motion control of a non-holonomic wheeled robot. Also GP has been used as a tool for finding a behavioral controller for an autonomous robot as can be seen in Ebner and Zell (1999) where the application was a large service robot.

Topology Design The use of EC has not been restricted to just evolving the controllers for behavior of the autonomous robots, but also for finding the topology such that the controllers could better exploit beneficial designs. One of the approaches in which topology was also evolved can be seen in Lee, Hallam, and Lund (1996) where a GA was used to evolve the topology which was followed by the application of GP for evolving a behavioral controller for the evolved robot. Differently, in Mautner and Belew (1999), a GA was used to evolve both the topology and the controller, which in this case consisted of a Neural Network of an autonomous robot. In Dittrich, Skusa, Banzhaf, and Kantschik (1999), EC was not used directly for the purpose of evolving the structure of a robot, but GP was used to find a controller for a robot with random morphology.

Subsystem Control There are also examples where EC has been used to find a controller for a specific part of the behavior of an autonomous robot. This is usually the case when it comes to movement of non-wheeled robots. One of the exceptions to this can be found in Kim and Shim (1995) where EP was used to find a robust locomotion controller for an autonomous robot with limits on velocity and acceleration. Otherwise, Hondo, Nishikawa, Yokoi, and Kakazu (1998) used GP for finding a locomotion controller for a starfish robot, and Parker and Mills (1999) with followup work in Parker (2001) used a GA to find an adaptive gait controller for a hexapod robot. Also, Rodrigues, Prado, Tavares, da Silva, and Rosa (1996) used a GA to evolve a controller for locomotion of a biped robot.

Evolvable Hardware The use of EC for controller design is not limited to software-based solutions. There also exist hardware-based approaches to finding controllers. This area is known as evolvable hardware and is usually based on the possibilities generated by field programmable gate arrays. The controllers found using evolvable hardware are usually very specific with regard to the system it is designed for, but this also allows the evolved controller to take advantage of the structure. This was evident in Thompson (1995) where an asynchronous controller was designed for a wall-avoiding robot.

Multi-Agent Systems The previously mentioned examples regarding autonomous robots were only concerned with the behavior of an isolated robot. One example of controller design for autonomous robots with multiple agents in mind can be found in Watson (1994). Similarly, in Mikami, Mitsuo, and Kakazu (1996), Agah (1996), and Louis and Li (1997) GAs were used

for evolving controllers that would be suitable for behavioral control of robots in a setting with multiple agents.

Test Scenarios A fitting end to this discussion of EC in the field of robot control can be found in Schultz, Grefenstette, and De Jong (1995). In that paper, a GA was used not to evolve a controller for an autonomous system, but for testing of intelligent controllers. It was an attempt to use EC as a test-generator, such that the performance of intelligent controllers could be examined in detail, especially for those cases where an unpredicted behavior might occur when certain fault scenarios were to happen. In this specific case, the approach was tested on a flight controller for landing on an aircraft carrier and an autonomous underwater vehicle.

This concludes a comprehensive, yet only scarce, discussion of the possibilities of using EC in the field of robotics, both autonomous and non-autonomous. Robotics is, however, not the only application area for which EC have been used, which should become apparent in the following section.

3.2.2 Other Applications

Filters The design of filters is an area under electrical engineering which is related to control engineering and also within this field has the use of EC proven useful. This can be seen in Neubauer (1997) where a GA was used for the design of analog IIR filters with variable time delays. The purpose for the designed filter was for optical control of microwave signal processors. Usually, filters are part of a larger system and in some cases, it can thus be necessary to be able to perform identification of these.

System Identification In order to design controllers for different systems, it is sometimes necessary to have a certain amount of knowledge of those systems. That is where the field of system identification comes in, and also this field of work has benefited from the application of EC. This can, among other things, be seen in Kristinsson (1989) and in the later work Kristinsson and Dumont (1991) where GAs are used for parameter identification of a system with a structure that is assumed known. By using the system identification, it was then possible to use a GA for optimizing the parameters of a controller, which was applied to an experimental setup of controlling the water level of a tank. Another time a GA was used for system identification was in Iwasaki, Miwa, and Matsui (1999) where it was used for identification of structured motion control systems using both one- and two-mass rigid systems. Differently in Chen and Kawaji (1999) it was actually the Probabilistic Incremental Program Evolution (PIPE) algorithm of Sałustowicz and Schmidhuber (1997) that was used for identification of a nonlinear system.

The area of system identification was just another example of how EC has been applied to different control applications. Another of these control applications is the issue of scheduling, which is described next.

Scheduling The issue of scheduling is based on the fact that there might be a limited number of resources available for different systems when they are running. Thus, a lot of work has gone into finding optimal solutions to the allocation of resources for a variety of occasions. Because the

scheduling problem in many cases cannot be solved analytically, there have been several attempts that have applied EC to the difficult task of resolving that issue. There are several very different problems belonging to the category of scheduling. One such problem is the flowshop scheduling of chemical processes described in Cartwright and Tuson (1994) where a GA was used to find the optimal topology of interconnected reactors such that throughput could be maximized. Similarly, in Tamaki, Sakakibara, Murao, and Kitamura (2002), a GA was used for solving a flexible shop scheduling problem. Another problem falling into the scheduling category is elevator group control where an optimal control strategy is sought, such that travel time, waiting time, and excess distance traveled can be minimized. This issue of elevator group control problem was raised in Alander, Ylinen, and Tyni (1995) where a distributed GA was used to solve the problem. Another related approach was used to solve the problem in Kim and Moon (2001) where a steady state GA in conjunction with a local search was used to find the optimal assignment strategy. Finally, a resource allocation problem was solved in Krishna and Naik (2000) using EP. That specific approach aimed at optimally controlling the resource allocation of semi-autonomous mission-critical distributed systems.

Most of the approaches so far have used EC as the primary tool for finding the optimal controller strategy/design. There are, however, also many approaches which have used a combination of EC and other control related optimization tools. Some examples of these hybrid approaches will be discussed in the next section.

3.3 Combined Approaches in Control

The use of EC in the field of control engineering has not been limited to pure EC based approaches. There are many examples of approaches which have included elements taken from more conventional control engineering or other related fields. Lately, fuzzy logic and neural networks have been very popular fields of research within the field of control engineering and it is then logical that there have been many approaches which attempt to combine these areas with EC, which will be the main focus in the following sections. However, even though the majority of hybrid EC approaches include either fuzzy logic, neural networks, or both, there have also been other hybrids. One of these hybrids can be found in Chen, Parmee, and Gane (1997) where a GA was combined with successive linear programming to solve a mixed integer non-linear optimization problem. This was then later used to design topology and control strategies for both a power and a water system. Now, because there have also been quite a few attempts made to compare different control methods with each other, this issue will also be discussed in one of the sections to follow.

However, to begin with, the next section will give some examples of hybrid approaches combining fuzzy logic and EC.

3.3.1 Fuzzy Control

Using EC as a tool to develop rule based control schemes have been used quite extensively since the late 1980s. One reason for this, might be due to the similarities between classifier systems and fuzzy logic systems. Having given that statement, it is most natural to give a short introduction to the field of fuzzy systems. The basic concept of a fuzzy system is shown in figure 3.1.

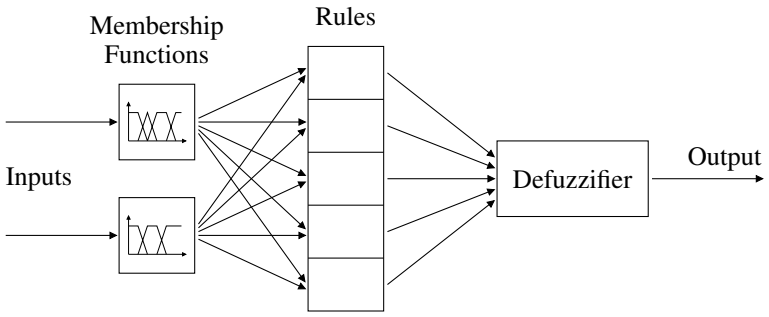


Figure 3.1: A general fuzzy based system uses membership functions to transfer the input into fuzzy logic expressions which are then used in the rule base to generate an output through a defuzzifier.

It can be seen that the input is fed into a set of membership functions that transform the input into fuzzy expressions. These expressions are then used as input to a rule base that, based on different input, will give different outputs. Finally, the outputs from the rule base are piped through a defuzzifier, which generates the actual output of the system based on the fuzzy expression generated by the rule base.

It should thus be clear how a fuzzy system has strong similarities with an LCS as described in section 2.5.6.5 on page 28. Quite interestingly, many other approaches than classifier systems have actually been used for finding fuzzy logic and other rule based controllers. One of the approaches that has been used most widely for this purpose is GAs.

Genetic Algorithms Kuchinski (1985), Odetayo and McGregor (1989), and McGregor, Odetayo, and Dasgupta (1992) are examples where GAs have been used for optimizing control rules for different systems. In the latter two papers, the system under consideration was based on the very popular cart-pole balancing problem. This problem is particularly interesting because it is used as an example for so many different rule based systems. An illustration of the cart-pole problem is given in figure 3.2.

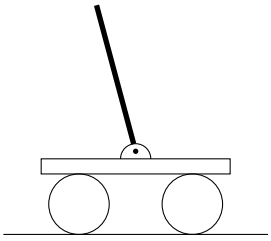


Figure 3.2: The cart pole balancing problem is an attempt to move the cart in such a way that the pole hinged to the cart is balanced in an upright position.

The main focus of the cart-pole balancing problem is to move the cart in such a way that the pole, which is hinged to the cart, can be balanced in an upright position. Other approaches which have used the cart-pole balancing problem as an example are Kawaji, Ogasawara, and Honda (1994) and Thierens and Vercauteren (1991), the latter being a refined version of the approach originally used in Odetayo and McGregor (1989) by means of an algorithm known as GENITOR (Whitley, 1989).

When it comes to approaches that specifically use EC for optimization of Fuzzy Logic Controllers (FLCs), there are examples like Park, Cho, and Cha (1995), Kacprzyk (1995), and Kacprzyk (1996). However, there are quite a few different approaches in which EC have been used for optimization of fuzzy based systems. For instance, in Kundu, Kawata, and Watanabe (1995), a GA was applied to only optimize the rule base which is similar to the previous mentioned examples. There are, however, also quite a few applications in which a GA has been used to only optimize the membership functions (Ortega & Giron-Sierra, 1995; Yoon, Hwang, & Park, 1998). In other approaches, such as Li, Tsang, Rad, and Chow (1999) it has been the input scaling factors that have been optimized using a GA, while Zhao, Collins, and Dunlap (2003) have used a real-valued GA for optimization of both membership functions and scaling factors.

Most of the approaches which have used EC for optimization of FLCs have tried to utilize the evolutionary approach as much as possible such that the entire knowledge base could be optimized. This is evident in Kinzel, Klawonn, and Kruse (1994), Freisleben and Strelen (1995), Alander, Moghadampour, and Törmänen (1997), Hwang and Zein-Sabatto (1997), Herrera, Lozano, and Verdegay (1998), and Lin and Lin (1998) where both membership functions and rule base have been optimized using GAs.

The previously mentioned applications of EC for optimization of fuzzy systems have mainly used a regular GA in one way or the other. There are, however, also examples where other versions of GAs have been used.

Modified Genetic Algorithms In Hoffmann and Pfister (1996) and Hoffman (1997) it was the messy GA from Goldberg, Korb, and Deb (1989) that was used to evolve a Michigan-style rule base for an FLC while an ES was run in parallel to optimize real valued parameters. Differently, in Romzi, Nishino, Odaka, and Ogura (1999), it was a transformation type GA which was used for the optimization, whereas Karr (1991) used both a simple GA for offline and a μGA for online optimization of an FLC for the cart-pole balancing problem. However, as previously mentioned there have also been quite a few examples of using LCS for finding FLCs.

Learning Classifier Systems Naturally, the use of classifier systems for optimization of fuzzy logic controllers have also been attempted. Among these classifier systems approaches, there have been examples which have used only a Pittsburgh-style classifier (Carse & Fogarty, 1994; Carse, Fogarty, & Munro, 1995), only a Michigan-style classifier (Velasco, 1998), or both approaches (Velasco & Magdalena, 1995; Pipe & Carse, 2000). The applications used in those approaches varies from adaptive distributed routing of package switching networks, over control of a mobile robot, to control of a steel rollmill. Also, the field of reinforcement learning, which is closely related to the classifier systems approaches, have used evolutionary approaches. This can be seen in Fogarty (1994a) where once again the suggested approach is applied to the cart-pole balancing problem. Of course there have also been other examples of using EC for finding FLCs.

Other Methods Some of the other methods include the EP based approaches of Kwon, Won, and Lee (1998), Kim and Jeon (1996), and Chellapilla (1998), but there are also examples such as Koza and Keane (1990), Alba, Cotta, and Troya (1996), and Yamazaki, Kundu, and Hamano (1998) which have used GP for optimization of FLCs.

Further, the use of online EC based approaches for fuzzy logic control has on several occasions been attempted by Karr in his work for the U.S. Bureau of Mines (Karr, Meredith, & Stanley, 1990; Karr, 1991; Karr, 1992; Karr & Gentry, 1993), and also the adaptive sliding mode controller has been augmented with the use of fuzzy logic and EC (Lin & Chen, 1995).

Finishing up the applications of EC for optimization of fuzzy systems, there are examples such as Urbančič and Bratko (1992), which discuss how knowledge of a qualitative model of a system can be used to generate control rules for automatically synthesizing a control strategy for a dynamic system. Another modeling approach is found in KrishnaKumar and Satyadas (1995) where multiple fuzzy models are evolved using clustering niches for an aircraft control problem.

With this extensive description of how EC has been used in conjunction with fuzzy based systems/controllers, it is natural to address another field which have used EC extensively for optimization purposes, namely the field of neural networks.

3.3.2 Neural Networks

In the field of Neural Networks (NNs) there are several issues, which over time have proven to be quite difficult to solve. This includes the structural design of the network and also the optimization of the weights used. First, let us take a look at some of the approaches in which EC has been used to optimize the weights of the NNs, also known as training.

Optimization of Weights An example of a simple NN structure is shown in figure 3.3.

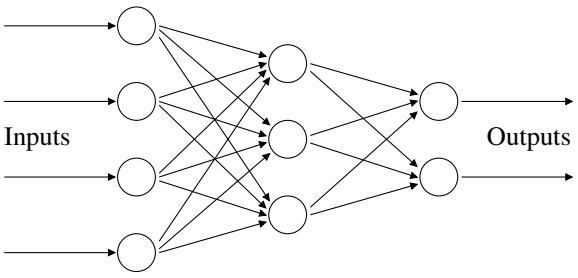


Figure 3.3: In this figure a simple feed-forward neural network is illustrated having 4 inputs, 4 input neurons, 3 hidden layer neurons, 2 output layer neurons and 2 outputs. The interconnecting lines represent the weightings throughout the network.

When optimizing the weights, it is thus the level of interconnectivity in the network which is manipulated, such that specific inputs will result in corresponding specific outputs.

The optimization of the weights has been the primary issue in papers such as Jones (1995) where

a GA was used to optimize the weights of an NN based PID controller for a non-linear system. In Schoenauer and Ronald (1994), the idea was to use a GA to optimize the weights for a NN based truck backer upper controller. This work was continued in Ronald and Schoenauer (1994) with the exception that the problem in this case was changed to the somewhat easier control of a lunar lander. Another GA based approach for optimization of the weights in an NN can be found in Whiteson, Kohl, Miikkulainen, and Stone (2003) where a neuro-evolutionary approach was used to train robots for the keepaway soccer problem. When considering other approaches than GAs, there are examples such as Saravanan and Fogel (1994) which used EP for optimization of the weights, or Salomon (1997) and Santos, Duro, Becerra, Crespo, and Bellas (2000) which took use of ES for optimization of the weights for NN based control of autonomous robots. However, as mentioned, there have also been attempts to use EC for evolving the structure of NNs.

Optimization of Structure When it comes to using EC for more than just optimization of the weights of NN based controllers, the usual approach is to use EC to evolve both the structure and the corresponding weights. An ambitious attempt was actually done in Cliff, Harvey, and Husbands (1993) and Harvey, Husbands, and Cliff (1993) which used the SAGA artificial evolution technique of Harvey (1992) to not only evolve the NN controllers for visually guided robots, but concurrently evolved the sensing morphology of the robots as well. A less ambitious approach where a GA was just used for finding topology and corresponding weights for a NN control of a wall following robot can be found in Law and Miikkulainen (1994). Also, Abu-Alola and Gough (1995) used a GA to train a NN and showed that the approach was capable of simplifying the network, since it was able to reduce the number of neurons in the hidden layer. One application which did not use a GA to optimize an NN can be seen in Morimoto, Baerdemaeker, and Hashimoto (1997). There an NN was used to model the response of fruit when it was being stored and this information was then used in a GA for finding the optimal set points for the control of the relative humidity in the fruit storing process. As this latter example showed, there are differences in how the combination of EC and NNs are performed. This is also the case for the next couple of examples, which used a more complicated interconnection of EC and NNs.

Hybrid Approaches There are also examples of different EC based methods for design of optimal NN controllers. One such example is Ohno and Furuhashi (1999) that used a GA to find the topology of a NN controller and EP was then applied afterwards to train it. The systems, which the NN was designed to control, were modified versions of the cart-pole balancing problem that included two poles. One version had the two poles attached to the cart, whereas another had only one pole attached to the cart while the other pole was attached to the end of the first pole thus making it a double jointed pole. Yet another approach, which only considered the original cart-pole balancing problem, used a tree base cellular encoding similar to GP for NN controller design (Pratt, 1994).

Other NN hybrids using EC for optimization include Zitar and Hassoun (1993), which used a classifier system in combination with a GA to generate macro rules that in turn was used to train the final NN controller of a truck backer upper problem. Also, a semi-online hybrid algorithm was used in Topalov, Kim, Kim, and Lee (1996) for tuning of an NN based PID-controller where a GA was used to find a near optimal solution which was then followed by a back-propagated NN for final online tuning. Interestingly, a hybrid approach using a GA, a backpropagation NN, and a multi-objective GA was proposed in Sette, Boullart, and Van Langenhove (1998). This

combination is quite rare, and the procedure for this approach was to use the GA to find the topology of the NN, which in turn was used in combination with the multi-objective GA to find a set of optimal control strategies for an industrial spinning production process.

The above examples are just a small selection of the work done for hybrid NN and EC based controller design. However, the hybrid EC approaches are not limited to NN or FLC. There are also a few examples which have tried to combine all three fields for the purpose of designing controllers.

3.3.3 Combined Fuzzy and Neural Networks

One example where EC is combined with both NN and fuzzy logic for controller design can be found in Pham and Karaboga (1999). There, a recurrent NN was used online in conjunction with a GA to identify the system model with the proper weights. The resulting best NN controller was then chosen as input for a FLC. This controller was then tested on both linear and non-linear systems with noise and parameter variations and proved to be quite adaptive in for all of those cases.

There have been quite a few success stories of how EC could be used in design of controllers. Along with this, there are also some papers, which have performed specific comparisons of different EC and non-EC based approaches.

3.3.4 Comparative Studies

One of the comparisons between EC and non-EC based approaches can be found in Searson, Willis, and Montague (1998). Here, a controller evolved using GP was compared with known optimal PID controllers for two types of systems, an Auto-Regressive eXogeneous (ARX) system and a continuous stirred tank reactor system. The conclusion of this paper was that GP could only find near optimal controllers which could not perform well in untrained situations. The GP approach also had the drawback that stability of the controller could not be built into the algorithm itself. Another comparison of EC and non-EC based methods was performed by Fukunaga, Marks, and Ngo (1994). A comparison was made of different GA, EP and Stochastic Hill Climbing (SHC) techniques for finding an optimal banked stimulus response controller for motion synthesis of 2D animated figures. Due to the simplicity of the problem, it turned out that the method that performed best was SHC applied to a population. Comparisons have also been done with NN and fuzzy logic systems. In Chiaberge, Merelo, Reyneri, Prieto, and Zocca (1994), a comparison of a pure NN, a pure GA, a combined NN and GA, a pure Simulated Annealing (SA), and a hybrid GA+SA was performed for tuning of a PID controller of a one-axis magnetic bearing. While the pure GA approach proved to be best, the authors preferred the combined GA+SA since it had performance near that of the pure GA while having a low memory requirement due to the SA. The preferred approach thus conformed well with the intended implementation using analog silicon. A comparison of finding an optimal controller for the flexible cart-pole balancing problem using a pure NN, a pure fuzzy logic, and a combined GA fuzzy logic approach was performed in Dadios and Williams (1998) with some of the details further explained in Dadios and Williams (1996). In this case, the GA based approach was found to be optimal since it did not require any prior information of the system to find the controller, whereas the other approaches needed information

of the system dynamics before a successful controller could be found.

Besides the many previous examples, there are also some publications which have addressed a range of the different areas discussed previously. For further information on the possibilities of using EC based approaches for controller design, the reader is referred to Renders, Nordvik, and Bersini (1992), Fleming and Fonseca (1993), and McDonnell (1997). Some volumes, like Biondo and Drummond (1994), Herrera and Verdegay (1996), and Jamshidi, dos Santos Coelho, Krohling, and Fleming (2002) also include comprehensive knowledge about NN and fuzzy systems and the combination of these with EC-based optimization.

In addition to the many EC-based approaches discussed previously, there are also other methods that have been inspired by nature, but do not fall under the general EA description. Since these methods also have been used for control related optimization, a short introduction to those areas and the areas for which they have been used will be given.

3.4 Related Methods

In the following, some methods will be described which do not fall under the category of EC, but which still have their origins in nature. As such, the methods do not fit into the general EA framework given in section 2.4 on page 9, but nevertheless they have been used for controller design and they deserve to be mentioned. One of those methods is particle swarm optimization.

Particle Swarm Optimization Particle Swarm Optimization (PSO) is based on the principle that a flock of birds (a swarm) is usually better at finding food than a single bird. As such, the method uses a population of individuals just like in EC. In PSO, however, the individuals change position in the search space based on a current position and speed and also on the knowledge of a personal best value and the global best value of the swarm.

The uses PSO have seen so far in the field of control engineering include reactive power control in electric power systems (Fukuyama, Takayama, Nakanishi, & Yoshida, 1999), optimization of NN weights for control of a bioreactor (Conradie, Miikkulainen, & Aldrich, 2002), and model predictive control of the temperature in a green house (Coelho, de Moura Oliveira, & Cunha, 2002).

Another method that does not fit under the general EA framework is the concept of immune algorithms.

Immune Algorithm Immune algorithms are inspired by the immune systems found in living creatures and how they are able to adapt to different situations. This has lead to an investigation of how immune systems can be represented and simulated on a computer which have mainly taken place within the last two decades (Forrest & Perelson, 1991; Forrest, Smith, Javornik, & Perelson, 1993; Hightower, Forrest, & Perelson, 1995; Oprea & Forrest, 1999; Hofmeyr & Forrest, 1999). As such, an algorithm based on the foundations of the immune system has been proposed. The Immune Algorithm (IA) is thus based on how the immune system of living creatures work. It works in three stages:

- Generation of Diversity

- Establishment of Self-Tolerance
- Memory of Non-Self

This allows the IA to be insensitive to known patterns (self) and instead be sensitive with regard to unknown patterns (non-self). For instance, in Ishida and Adachi (1996) such an IA was used for active noise control. The IA was good at suppressing the disturbance when it was first encountered but the power of the method was most emphasized by the fact that when the same disturbance was encountered later it was suppressed much better than at the first encounter. Thus, the IA seemed able to mimic the workings of a true immune system. This could, as the authors also point out, easily be applied to the harder problems of fault diagnosis and dynamic scheduling.

So far, among the vast amount of examples where EC have been used for control engineering purposes, there have been relatively few cases where a multi-objective algorithm have been used. Among the examples discussed in this chapter so far, there have been only one that used multi-objective optimization. As such, the time has finally come to discussing the importance of multi-objective algorithms and thus the use of multiple fitness function within the field of control engineering.

3.5 Multi-Objective Approaches

The reason why the use of MOEAs for control purposes have been so limited might be related to the fact that the field is relatively new. However, many of the issues encountered when designing controllers often require that multiple objectives are optimized, which makes MOEAs with their multiple fitness functions the most obvious choice for solving those problems.

Some of the earliest approaches to using MOEAs for control purposes use the MOGA from Fonseca and Fleming (1993a). The applications that MOGA have been used for covers optimization of the response of a Pegasus gas turbine engine (Fonseca & Fleming, 1993b), finding noise sensitivity tradeoffs for a double integrator plant with excess phase (Fonseca & Fleming, 1994), and design of both linear and nonlinear controllers (Kundu & Kawata, 1996; Kundu, Kawata, & Watanabe, 1996). Also, in Fonseca (1995) the use of MOGA was more extensively discussed along with the application of it to mixed H_2/H_∞ controller design, controller design for a Pegasus gas turbine engine, and nonlinear system identification.

There have also been approaches that considered multiple objectives for optimization of controllers, but which cannot be considered as true multi-objective optimization problems since they were implemented using only a single fitness function. These are the so-called mixed multi-objective, or single fitness, approaches where each objective is assigned a weight and a sum is taken over all objectives thus resulting in a single fitness value. Some examples of this weighted sum approach are given in Coello Coello, Christiansen, and Aguirre (1995) and Fang, Kellogg, Conlan, Dickerson, and Cook (2003). Further, in Pedersen, Langballe, and Wiśniewski (2002), another method of weighting was used for controller synthesis of a mixed H_2/H_∞ controller.

With these examples of multi-objective approaches, the discussion of how EC have been applied to applications within the field of control engineering has come to an end. The chapter will, however, be concluded with some remarks regarding many of the approaches discussed so far and how improvements to those approaches could be obtained.

3.6 Concluding Remarks

This chapter has been meant as a comprehensive discussion of how EC, which is a relatively unknown field to many control engineers, has been used quite extensively for controller design over the last couple of decades. With this said, a large number of the papers mentioned in this chapter unfortunately suffers under the lack of theory that have only come forward within the last few years. This is not meant as critique towards the papers mentioned, but as an encouragement to revisit some of the applications with the new theoretic tools, such that results that might be even better than those previously found, could be obtained. For many of the simple approaches, the choice of different parameters had been based on specific values found to be good in other papers, but which might not have been optimal for the application for which they were actually used.

It is also very clear that many of the approaches have used only a single fitness function, which in retrospect might seem insufficient based on the inherent multi-objective nature that surrounds many controller optimization and design issues.

There are, however, still many issues which have not yet been investigated sufficiently such that EC can live up to its full potential. The remainder of this thesis will try to bring up some of the salient issues in the field of multi-objective optimization such that it will become possible to use EC based approaches for controller design in more intelligent ways and more autonomously.

Chapter 4

Constraints and Objectives

In order to fully take advantage of EC for control engineering purposes, it is important to emphasize on the relevant issues, while keeping the details of the underlying less relevant information to a minimum. If EC is ever to be used as an efficient optimization tool for control engineers, it should not be necessary to have detailed knowledge of all the different tweaks and tricks used in conjunction with the different algorithms. In essence, the need for substantial introductions to the field of EC, more than the one given in chapter 2 on page 5, should not be a requirement in order to use the algorithms. Unfortunately, there are still a range of issues that suffer from lack of understanding in order for EC to become an efficient optimization tool for control engineers. In spite of these shortcomings, and even if they some day are sufficiently resolved, there is one subject that would still be necessary for control engineers to have an understanding of before using EC for optimization purposes. This subject is what will be discussed in this chapter, namely the issue of how the fitness functions should be formulated based on the desired objectives and constraints.

In order for an efficient black box optimization toolbox based on EC some day can be designed, it is important to have an understanding of how different objectives and constraints must be formulated into fitness functions in order to obtain good results. Formulating the fitness functions incorrectly could result in poor results, since any loophole would surely be exploited by the algorithm, thus giving an unusable result. One of the major issues in this regard of designing fitness functions is to know the difference between an objective and a constraint and how the different algorithms treat these. The discussion in the next section will first focus on a description of objectives and their formulation which will be followed by a section in which the issue of constraints and their formulation are discussed.

4.1 Objectives

An objective is basically an expression that is sought optimized. This means that it is a measure which is sought to be optimized in order to obtain an optimal level of performance for the system under consideration. If, for instance, an objective is to minimize the output error of a given system,

then it is easy to tell if one solution is better than another based on the error obtained when using the two solutions. So, for an objective it is possible to always tell if one solution is better than another at reaching the objective and it is always the goal to find the optimum value.

All EC based algorithms contain one or more objectives. Without an objective, there would be no need for the algorithms at all. However, the way the objectives are implemented in the different algorithms differs quite a lot. The main difference is most clear when it comes to comparing algorithms with a single fitness function and those containing multiple fitness functions. This is why the two cases will be discussed separately, starting with the single fitness case.

4.1.1 Single Fitness Case

So far, most of the algorithms in existence based on EC have only had one fitness function. This does not mean that those algorithms only tried to optimize one objective, but that the objectives were always reformulated into a single fitness function. Naturally, this leads to the question whether such an approach is advantageous or not. It is easy to realize that if there is only one fitness function to optimize then it is easy to determine whether the fitness value of one individual is better than that of another by just comparing the fitness values. As such, it is easy to make decisions for this one dimensional problem.

Amongst the problems for which EC have been used there have been many cases in which several objectives have been combined into one fitness function using a weighted sum. This is one way of converting a problem using multiple objectives into a single fitness problem. However, the fact that it requires weights means that they must be specified beforehand which can be quite difficult. In fact, the best chance of setting the optimal weights could easily require an EA to be used, thus complicating matters even further.

However, even problems that only have one fitness function are not necessarily easy to solve. As mentioned in section 2.5.6.4 on page 27, the issue of multimodality can cause algorithms with a single fitness function a considerable amount of problems. As previously mentioned, this problem of multimodality can be directly transferred to some of the issues encountered for multi-objective algorithms. Luckily, most regular problems having a single fitness function do not necessarily encounter the multimodality problem, but for those who do encounter that problem there have been different methods that tried to compensate for the difficulties. Some of these methods along with the issue of genetic drift are what will be shortly addressed next.

4.1.1.1 Multimodality and Genetic Drift

As mentioned in section 2.5.6.4 on page 27, the drifting issue in connection with multimodality was investigated in Goldberg and Segrest (1987) and continued in Goldberg (1989) and Goldberg (2002). In these publications, methods such as the gamblers ruin models and also Markov chains were used to estimate the time needed for the population to converge into identical individuals when there was no difference in the fitness values of the individuals. The case where all individuals have identical fitness values actually corresponds to a scenario where the selection pressure is 0. Both the simple, yet good estimate of the gamblers ruin model and the more exact estimate given by the Markov chains showed the effect of the drifting issue on the evolutionary process and that a niching method was needed in order to deal with that problem.

Actually, the niching is an artificial way to, ideally, obtain a full ordering of the individuals, such that it will be possible to distinguish the individuals from each other and make good decisions. For the case with one fitness function where each individual only have one fitness value, it is only possible to base the niching on the diversity of the individuals in the phenotypes or genotypes. As such, a preference will then be given to individuals which are non-similar in either genotype or phenotype. This will help the algorithm maintain a diverse set of phenotypes, but it also means that the niches, in which the optimal values are, must be maintained. Having a population which is too small with regard to the number of niches that must be maintained could thus result in the loss of some of these niches. On the contrary, choosing the population much larger than is necessary would result in a lot of wasted computations.

So, even though there exist a wide variety of different single objective algorithms, the approach in which a single fitness function is used is not without difficulties. It can thus be appropriate when using only one fitness function, if possible, to use a multi-objective algorithm, having multiple fitness functions, to solve a specific problem. This leads to a discussion of the strengths and weaknesses of using an approach based on multiple fitness functions.

4.1.2 Multiple Fitness Case

When it comes to algorithms using multiple fitness functions, the area which must be searched suddenly expands to multiple dimensions, growing for each fitness function. Even though this enlargement of the search space makes the problem more complicated to solve, it also provides some advantages. Not only can the optimization of multiple fitness functions help give an understanding of the structure of the problem, but it can also provide a wide set of Pareto-optimal solution possibilities to choose from.

In many cases it is possible to convert a problem having a single fitness function into a problem with multiple fitness functions by splitting it into several sub-problems or by assigning each of the perhaps multiple objectives with a fitness function of its own. This can help giving a better understanding of the problem while simultaneously obtaining a set of optimal solutions. It is, however, not all problems which will result in a set of Pareto optimal solutions. If all fitness functions of a problem are non-contradictory, it is possible that the usual set of optimal solutions will be reduced to consist of only a single individual. That specific situation is quite rare for real-world applications where a tradeoff between multiple fitness functions usually take place. However, upon realizing what is important for an algorithm, it becomes easier to understand some of the underlying issues.

The goal of MOEAs actually consists of two parts as mentioned in Deb (2001) and Coello Coello, Van Veldhuizen, and Lamont (2002), namely that the solutions found must be:

1. Close to the Pareto optimal front
2. Diverse

This is also illustrated in figure 4.1 on the next page where it is clear how solutions near the Pareto optimal front first are sought followed by a search for diversity along the front.

The first requirement can be obtained using the conventional concept of dominance and does not have a need for any niching or crowding measures. A good algorithm would thus be able to find a

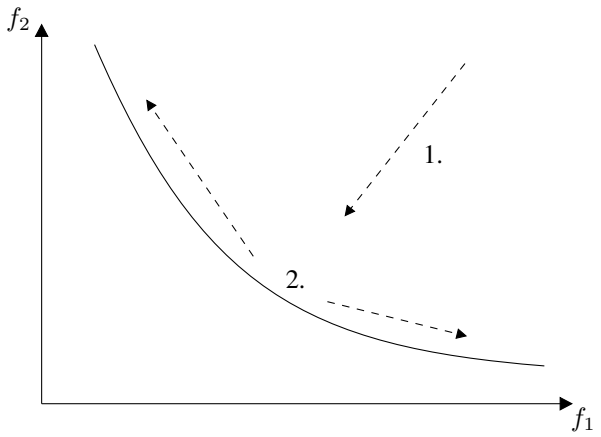


Figure 4.1: The goal for MOEAs is to find a diverse set of solutions near the Pareto optimal front by first finding the front and then creating diversity along it.

set of solutions as close to the Pareto optimal front as possible. However, the second requirement can be more difficult to obtain. In order to obtain a diverse set, it must be specified what can be considered as a set of diverse solutions, but it must also be understood how dominance has influenced the diversity of the solutions.

In section 2.10 on page 41 it was discussed how the concept of dominance could be applied to MOEAs and how this, unfortunately, led to the need for niching and crowding measures. It has been mentioned previously that the concept of multimodality is similar to the problems encountered in connection with the use of dominance for problems having multiple fitness functions. This plays an important role in the understanding of the MOEAs. When performing the ranking of different solutions based on dominance, the different individuals are classified to belong to a specific rank. This ranking is essentially a mapping of the individuals from genotype or phenotype to $\mathbb{N}^+$. The ranking procedure thus creates a multimodal problem on $\mathbb{N}^+$ since there usually are more than just one individual having the same rank. In effect, this mapping of multiple individuals to each rank is closely connected to the fact that the concept of dominance was not reflexive and not antisymmetric. In fact, dominance has converted the multiple fitness case into a single fitness case that is highly multimodal. Only for the case where there is only one individual belonging to each different rank, a situation arises that do not need any niching or crowding measures. However, as explained before, this happens very rarely for real-world applications for which controllers are usually designed.

The issue of obtaining some form of full ordering then becomes necessary in order to be able to obtain some level of diversity for the individuals belonging to the same rank. This leads to one of the areas where MOEAs are different from the single fitness case. Where, for a single fitness EA, it is genotypic/phenotypic diversity that is sought when faced with multimodality, for MOEAs it can be diversity of either genotypes/phenotypes or fitness values, or maybe even both that can be considered. The goal of maintaining a diverse set of solutions in MOEAs is mostly meant to apply to the fitness values, but the underlying genotypes/phenotypes can easily be different for identical

fitness values and as such, the issue of diversity with regard to the genotypes/phenotypes sometimes also has to be considered. Most MOEAs do, however, only consider the values in fitness space, since it is usually there the diversity is of interest. As such, it is more interesting to choose between two different solutions that give rise to different fitness values than to choose between two different solutions which give rise to the same fitness values. This does not mean that diversity of the genotypes/phenotypes for MOEAs should be ignored, since there could exist problems where this could become an issue. However, since it is most interesting to obtain diversity for the fitness values for the majority of problems, this is what will be focused on in this thesis as well.

Once crowding or niching have been applied, it becomes possible to obtain a diverse set of solutions through the selection operator. One remarkable thing about this is that it becomes possible to construct a mapping from $\mathbb{N}^+ \times \mathbb{R}_0^+$, which is the space spanned by the rank and crowding values of the individuals, to $\mathbb{R}^+$. Each individual is thus assigned a single value representing its rank and corresponding crowding value and the problem can then be considered as a problem having a single fitness function. This essentially means, that any problem having multiple fitness functions can be solved using an algorithm meant for solving problems with only a single fitness function. Thus, it is possible to use any single objective EA to solve any of the problems for which a MOEA using ranking and crowding can be applied. The main issue here is though, that the result will depend heavily on which ranking method is used and also how the crowding is calculated. Usually, it is still desirable to solve problems having multiple fitness functions using a MOEA, since the tradeoffs in the problem can be identified more easily and thus a better understanding of the problem can be obtained.

However, besides from the issues regarding the objectives, there are also several issues regarding constraints which must be kept in mind. These issues will be the focus of the next section.

4.2 Constraints

In contrast to an objective, a constraint is an expression that must be upheld. This means that by defining constraints it is indicated that a solution is not feasible or valid unless it meets with all of the constraints set forth. If faced with a choice, solutions which uphold all of the constraints set forth should be preferred over those who do not meet all constraints, and if no solution upholds all constraints, an emphasis should be put on those who are closest to upholding the constraints. A constraint should thus only be taken into consideration if it is not upheld. If a constraint is upheld, then it should have no influence on any further choices made, since there are no further level of choice possible. A constraint is either upheld or not. If it is necessary to distinguish further than such a level, it should then be considered as an objective as described above.

Constraints are used in most EC algorithms, either explicitly or implicitly. An example of implicit use of constraints can thus be seen in binary encoded algorithms. There, an interval is specified, thus placing a constraint on the range of the obtainable values for the individuals. This also shows that constraints not only can be applied to objectives, which is the usual case, but for phenotypes and even genotypes as well. The description of constraints given here will focus on the explicitly defined constraints. Even though constraints in a sense reduce the effective size of the search area, it is important to notice in what way this goal is actually achieved. First, let us take a look at how this is done for algorithms using a single fitness function.

4.2.1 Single Fitness Case

For the single objective algorithms using a single fitness function the method of implementing different constraints has been as a part of the fitness function (Ram, Arkin, Boone, & Pearce, 1994; Smith & Stonier, 1996; Pohlheim & Heißner, 1999). The way this has been achieved is for each of the constraints to add a penalizing term to the fitness function. Then, any individual, which do not satisfy one or more constraints, would receive a poor fitness and be less likely to be selected for survival.

An example of using such penalty terms in a control related setting would be the way Lagrange multipliers are used for various controller design purposes. For such a situation the Lagrange multipliers thus acts as constraints for finding an optimal controller.

In order to implement this method of penalty terms it is a requirement that some knowledge of the possible outcomes of the fitness function is known a priori. Now, if it happens that the penalty terms are set too low, it would be possible for many of the individuals which did not actually meet the constraints to survive, which could lead to an optimal result that did not meet the constraints set forth. Setting the penalizing term too high would reduce the effectiveness of the algorithm, since individuals that are very close to satisfying one or more constraints would not be able to influence the evolutionary process. This could then reduce the speed with which the algorithm could find good solutions that satisfy the constraints in the vicinity of any of the unsatisfied constraints. This can be illustrated using a simple example:

Example 4.1 (Constrained OneMax Problem) *The OneMax problem, widely used by Goldberg (2002), considers a binary string x and seeks to maximize the number of ones in that string. Thus, the fitness function can be written as*

$$f = \sum_{i=1}^{l} x_i \qquad , x_i \in \{0, 1\} \tag{4.1}$$

where x_i represents the individual bits and l is the total string length. The fitness function f is then sought maximized. However, let us consider the modified version in which the first element x_1 is required to be 0. A penalizing term of $2n$ can then be added to the fitness function yielding

$$f = 2l \cdot (1 - x_1) + \sum_{i=1}^{l} x_i \qquad , x_i \in \{0, 1\}. \tag{4.2}$$

It is seen that for any individual not meeting the constraint $(x_1 = 1)$, the penalty will not add to the fitness value whereas an individual meeting the constraint $(x_1 = 0)$ will get an additional bonus of $2l$ added to the fitness value. The penalty term thus penalizes any individual not meeting the constraint by not adding a bonus. The problem for $l = 4$ is illustrated in figure 4.2 on the facing page where the individuals are represented by their decimal counterparts.

Now, let us consider a small population of 4 individuals for the shown case where $l = 4$ (1111,0100,1011,0000). The 4 individuals have a corresponding fitness value of 3, 9, 2, and 8 respectively. It is clear that the first individual (1111) is the one which is closest to the optimal value (0111) since it would only require x_1 to change to a 1 in order to become optimal, but due to the penalty term it receives a very low fitness. The algorithm for this specific case would

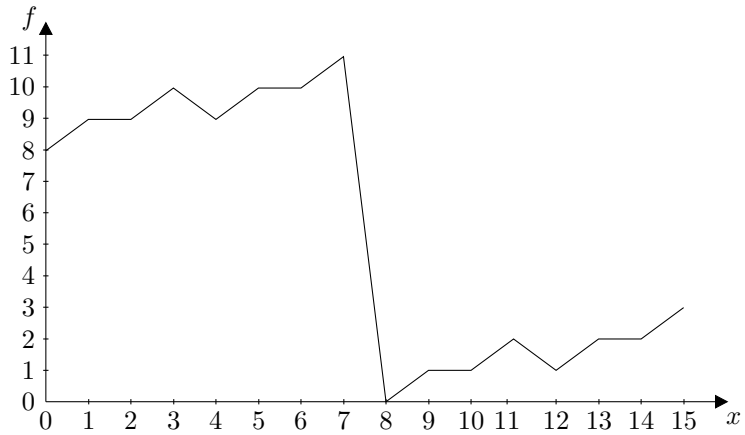


Figure 4.2: The constrained OneMax problem is illustrated using the decimal counterparts of the individuals along the x-axis and the corresponding fitness values on the y-axis.

*most likely loose the above average building blocks (*111) and (**11), thus requiring additional computation in order to regain them at a later time.*

The penalty term, which have been described above, can in effect be considered as a "prioritized" fitness contribution. The main focus is, however, that the penalizing term must have an optimal value for several different individuals. If not, then what was considered as a constraint would in fact be the only term on which optimization is performed, since it would dominate the fitness function. So, it is quite clear that when the constraint is implemented as a penalizing term, then the search area is not actually reduced, but merely warped in a way such that individuals that satisfy the constraints are emphasized. In some cases, the inclusion of penalty terms can introduce new or even move existing local extrema, but this does not pose a problem for most EAs since they can successfully handle such situations.

Now, the issue of applying constraints to the case having multiple fitness functions can be handled in a variety of ways, which will be further explained in the next section.

4.2.2 Multiple Fitness Case

This discussion of constraints for the multiple fitness case will focus on those MOEAs which use ranking schemes. However, even with the focus on ranking based MOEAs there are still many ways in which constraints can be implemented. A logical extension to implementing constraints for the multiple fitness case would be to emulate the procedure of the single fitness case and use penalty functions. By adding penalizing terms to the different fitness functions, it becomes possible to accommodate constraints. Doing so would, however, not be optimal, since the constraints would need to be included in all fitness functions in order to ensure that they are satisfied, but at the

cost of introducing an unnecessary amount of extra computations. Also, since the multiple fitness approach allows for constraints to be either included more or less as constraints or as objectives, one of these cases are usually preferred.

It is, however, important to note the difference between a constraint being implemented as a constraint or an objective. Either way of implementing constraints results in the addition of extra fitness functions. It is, however, the relative importance of the fitness functions which will be the issue. Because of the ranking, a constraint will always be prioritized above any objective. As such, the constraints actually becomes highly prioritized fitness functions, which aid in restricting the search space. On the other hand, if a constraint is implemented as a regular objective using a fitness function, it would have to compete on equal terms with the remaining fitness functions, which eventually could result in some solutions being included in the Pareto optimal set even though they violate one or more constraints. The advantage of that approach would be that individuals which do not meet the constraints could help guide the evolution in the right direction, thus overcoming the drawbacks discussed in example 4.1 on page 78. Thus, implementing a constraint as a regular fitness function rather than as a highly prioritized fitness function, could help the evolutionary process in the short run, but in the long run it would slow down the evolution since the individuals which do not meet the constraint would still be able to survive to a certain extent, and also be kept alive by the algorithm. This means that in most cases constraints are implemented as highly prioritized fitness functions and not as regular fitness functions.

Another key issue in handling constraints for multi-objective algorithms is how the ranking is performed when several constraints may be violated. In this case, there are different scenarios such as preferring individuals which violate the least number of constraints or preferring individuals which are closer to satisfying all constraints as done in NSGA-II. Another way might even be to prefer those individuals which are located in less crowded regions similar to the crowding methods used for regular fitness functions. As can be seen in Deb (2001), there have been several different suggestions to how constraints can be handled. Which constraint handling method that turns out to be optimal will most likely depend on the problem being solved. Because of that and since the discussion of constraints will be more relevant in a discussion of prioritized fitness functions, which is not the focus of this thesis, the issue of constraints will not be addressed further in the remainder of this thesis. The only issue with regard to constraints that will be mentioned further is the fact that they must be treated carefully.

When the constraints are implemented as highly prioritized fitness functions, they put a restriction on the search space. This does not mean that they reduce the search space, but that they emphasize evolution to occur in the unconstrained areas of the search space. However, one of the bigger issues regarding constraints actually consist of investigating to which degree constraints can be considered as an aid for the evolutionary process. Because just as constraints can be considered as an aid, they can also be considered as nuisances. This is because constraints that are difficult to satisfy will cause the algorithm to focus on these constraints only, thus more or less disregarding the regular fitness functions. The focus is then shifted to finding unconstrained solutions and, depending on the shape of the fitness landscape, this can result in not finding any Pareto optimal solutions. These considerations are important to keep in mind when formulating constraints for different problems that are to be solved using MOEAs.

With this comprehensive discussion of the issues regarding constraints and objectives for problems having either a single or multiple fitness functions, the time has come to sum up on the issues and identify the areas where further investigation is necessary.

4.3 Concluding Remarks

Throughout this chapter it has been discussed which issues are necessary to keep in mind when setting out to solve a difficult problem using EAs. In the later years, many successful attempts have been made to make a parameterless EA, but even for these parameterless EAs, that do not need any tuning of the different parameters, it is still necessary to formulate the constraints and objectives into fitness functions such that they conform with the problem while still keeping the problem solvable. Based on the considerations of this chapter, it should be clear that much care must be taken when making these formulations.

It was not only stated how it was possible to make conversions between regular fitness functions based on objectives and highly prioritized fitness functions based on constraints, but also how a problem having multiple fitness functions could be converted into a problem having only a single fitness function and vice versa.

Besides from the issue of priorities for fitness functions there is an issue that also has an important role in the way the evolutionary approach will perform. This is the issue of niching and crowding. The crowding must be performed in a way such that the algorithm will yield satisfactory results to the problem. However, the question is whether it is possible to apply the crowding in such a way that it will provide these good results for different problems without incorporating knowledge of the problem into the crowding mechanism itself, thus only basing it on a some form of user preference. This issue is what will be the main focus of the remainder of this thesis. By having a crowding mechanism that independently of the problem can provide a reasonably diverse set of solutions near the Pareto optimal front, the only remaining issue, would be to design the appropriate fitness functions based on the objectives and constraints, which varies from problem to problem.

Thus, a major hurdle before controllers can be designed more or less automatically based on a number of objectives and possible constraints is for the crowding mechanism to perform well, independently of the problem. This balancing act of performing the crowding calculations will be investigated further in the next chapters. The issues of priorities of fitness functions is thus left for further investigation at another time and the problems considered in the remainder of this thesis will only consist of regular fitness functions based on objectives with no constraints.

Chapter 5

Scaling Issues for Multiple Fitness Case

As concluded in the previous chapter, the formulation of the fitness functions must be carefully designed on a case to case basis. Since it is only the fitness functions that in principle vary from case to case, it should then be possible to create a broadly capable EA, where the formulation of the fitness functions is the only step necessary, and where the need for modifications to the algorithm itself are minimized. Some might say that an attempt to create such a generalized EA would result in a very inefficient algorithm, since it should be able to encompass a large variety of problems. However, by considering EAs as a mere tool for solving different optimization problems, it becomes necessary to minimize the need for parameters and other tweaks that require a comprehensive knowledge of EAs. Otherwise, it would only be researchers with a comprehensive knowledge of EAs that would be able to use the EAs efficiently for solving problems, and the use of EAs would thus not be available for those researchers that merely wanted an optimization tool to solve a specific problem. A generalized EA should thus only require the formulation of the fitness functions in order to produce usable results.

In this thesis, it will not be attempted to design a fully generalized EA from scratch. One of the main reasons for this is that there already exist good algorithms that are capable of delivering good results with only minimal EA specific input. Instead, this thesis will perform the logical step of investigating one of these existing good algorithms and modify it in such a way that the result will be the desired general EA.

The choice of algorithm to use as basis for this generalized EA has fallen on the Non-dominated Sorting Genetic Algorithm II (NSGA-II). One of the main reasons for choosing NSGA-II is because it is a multi-objective algorithm and because it, when it was proposed, was so successful that it quickly found use as comparison for other algorithms. There are also other successful multi-objective algorithms, such as Pareto Archived Evolution Strategy (PAES) (Knowles & Corne, 1999), Pareto Envelope-based Selection Algorithm (PESA) (Corne, Knowles, & Oates, 2000), Strength Pareto Evolutionary Algorithm (SPEA) (Zitzler & Thiele, 1998), and the successor Strength Pareto Evolutionary Algorithm 2 (SPEA2) (Zitzler, Laumanns, & Thiele, 2001).

However, other reasons why the choice fell on NSGA-II include its widespread use within different fields (Deb & Jain, 2002; Deb, 2002), the availability of the source code and also because of the extensive documentation of the algorithm itself in Deb, Pratap, and Moitra (2000), Deb (2001) and on the Kanpur Genetic Algorithm Laboratory (KanGAL) website (http://www.iitk.ac.in/kangal/).

Having chosen NSGA-II as a basis for creating a generalized version of an EA, the time has now come to investigate how well the algorithm performs for an arbitrary optimization problem.

5.1 Investigating NSGA-II

In order to test the algorithm on an arbitrary optimization problem, a simple control problem with conflicting objectives was devised. The problem was based on the system given in figure 5.1.

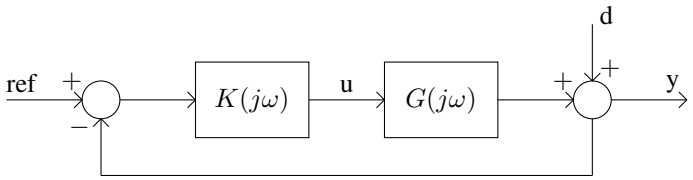


Figure 5.1: Overall structure of a simple control problem.

From the figure it is possible to formulate three different transfer functions that could be relevant for the design of a controller as listed below.

- Disturbance sensitivity (d → y): $f_{sens} = \frac{1}{1+K(j\omega)G(j\omega)}$
- Input sensitivity (d → u): $f_{inps} = \frac{K(j\omega)}{1+K(j\omega)G(j\omega)}$
- Closed loop (ref → y): $f_{cltf} = \frac{K(j\omega)G(j\omega)}{1+K(j\omega)G(j\omega)}$

Based on those transfer functions, the idea is to design several conflicting frequency characteristics. The algorithm will then be used to minimize the quadratic performance function for each of the conflicting frequency characteristics, and it will be seen whether the algorithm is capable of producing an acceptable set of tradeoffs for such a case. The plant model $G(j\omega)$ and the structure of the controller $K(j\omega)$ is chosen as follows:

$$G(j\omega) = \frac{1}{(j\omega)^2} \tag{5.1}$$

$$K(j\omega) = \frac{1}{x_1 + x_2 j\omega} \ , \tag{5.2}$$

where x_1 and x_2 are the values that can be used to create different characteristics of the frequency response for the different transfer functions. The desired values of x_1 and x_2 used as the reference in the different transfer functions $f_{sens_{ref}}$, $f_{inps_{ref}}$ and $f_{cltf_{ref}}$ can be seen in table 5.1 on the facing page.

Transfer function	Designation	x_1	x_2
Disturbance sensitivity	$f_{sens_{ref}}$	8	0.1
Input sensitivity	$f_{inps_{ref}}$	0.6	0.2
Closed loop	$f_{cltf_{ref}}$	4	1

Table 5.1: The desired values for the three different transfer functions of the simple control problem.

It is then only a matter of implementing the fitness functions into the algorithm and running it. The most straightforward way of implementing the fitness functions is by calculating the integral squared error and minimizing it. The way this is done for the simple control problem is identical for all three transfer functions and shown in formula (5.3).

$$\min F_y(x) = \sum_{k=1}^{n} \left(\frac{f_y(j \cdot 10^{-2+4\frac{k}{n}}) - f_{y_{ref}}(j \cdot 10^{-2+4\frac{k}{n}})}{f_{y_{ref}}(j \cdot 10^{-2+4\frac{k}{n}})} \right)^2 , \tag{5.3}$$

where n is the number of discrete points over which the integral squared error is summed and $y \in \{sens, inps, cltf\}$ indicates which transfer function is used in the calculation. It should then be obvious that by choosing $n = 200$ the formula given will summarize the proportional error between the reference transfer function and the transfer function being optimized at 200 points evenly distributed over a logarithmic axis from a value of $\omega = 10^{-2}$ to $\omega = 10^2$. The reason why the points are evenly distributed on a logarithmic scale is because the fitness function otherwise would emphasize mostly on high frequencies. Additionally, the use of a normalized term to calculate the proportional error is due to the fact that the fitness functions otherwise would be biased toward the areas where a high gain of the frequency response was desired.

Because a proper visualization of three dimensional space on a two dimensional piece of paper is difficult to obtain, the simple control problem will be split into three different scenarios. The result of these scenarios should then give the reader some idea of the issues encountered when performing optimization for this simple control problem.

For all three of the experiments performed, the parameters used in the algorithm were kept identical and are summarized in table 5.2 on the next page.

In the table, all of the operator specific values are those used per default in the NSGA-II algorithm. The only non-standard values include the range of the variables, maximum number of generations, the random seed, and the population size. The only interesting values of those are the maximum number of generations and the population size. The maximum number of generations is chosen at 200 to ensure that the algorithm will have time to both converge on the Pareto optimal front and to disperse the solutions such that a wide set of optimal solutions can be obtained. The population size is chosen to be 100 since it is expected that this will result in a nice coverage of the Pareto optimal front for the given problems. If the problem to be solved was not just a rough investigation of the given algorithm, it would be prudent to choose the population size carefully depending on the difficulty of the problem and maybe even use a population sizing mechanism similar to that given in Goldberg (2002).

The results obtained for running the algorithm with the specified parameters on the three combi-

Parameter description	Type	Value	Designation
Number of fitness functions	Functions	2	-
Number of variables	Reals	2	x_1, x_2
x_1 variable	Real	$[0, 100]$	$[x_{1_{min}}, x_{1_{max}}]$
x_2 variable	Real	$[0, 100]$	$[x_{2_{min}}, x_{2_{max}}]$
Selection operator	Tournament	2	s_{size}
Crossover operator	SBX	10	-
Crossover probability	-	0.9	p_c
Mutation operator	Polynomial	50	-
Mutation probability	-	0.5	p_m
Population size	-	100	N
Maximum generations	-	200	τ_{EA}
Random seed	-	0.1234	r_{seed}

Table 5.2: The parameters used in NSGA-II for the three examples of the simple control problem.

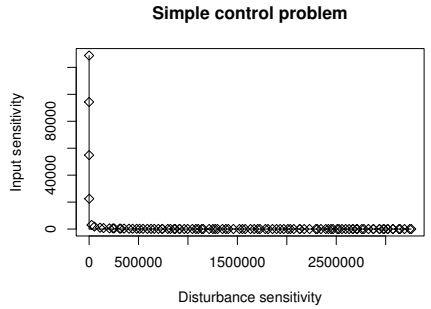
nations of fitness functions for the simple control problem are shown in figure 5.2 on the facing page[1].

From the figures, it is quite clear that the three different fitness functions F_{sens}, F_{inps} and F_{cltf} are weighted differently by the algorithm. In order of importance, the algorithm tends to bias mostly towards the disturbance sensitivity function F_{sens} followed by the input sensitivity function F_{inps} and with the least importance put on the closed loop transfer function F_{cltf}. The result of this bias is that the set of solutions is not as evenly distributed as could have been expected. Especially since there was not specified any information into the algorithm with regard to relative importance of the different fitness functions, it is quite noteworthy that the algorithm itself introduced such a bias. As seen in the figures, the algorithm tends to bias the solutions towards the fitness function having the largest values and the reason for this will be given shortly. First, however, it can be concluded that the algorithm was not capable of providing a diverse set of solution to an arbitrarily chosen optimization problem and it is thus necessary to investigate this issue further before NSGA-II can be used for optimization of other problems who might have similar characteristics.

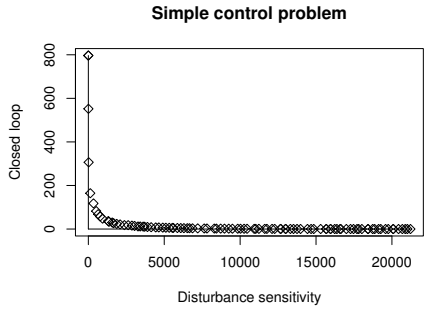
5.2 Explaining the Bias of NSGA-II

As explained above, the NSGA-II algorithm was unable to provide a usable set of alternative solutions for an arbitrarily chosen problem. The explanation to the occurrence of this deficiency is the fact that the scaling for each of the different fitness functions was not comparable. The algorithm thus had problems when the fitness functions had disparate scaling. This issue was discussed in Pedersen and Goldberg (2004) along with a possible solution to the problem. However, because of

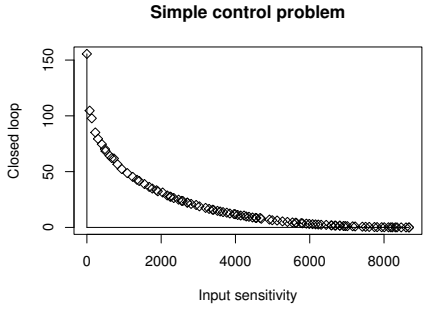
[1] All of the graphs presented in this thesis are based on a run with a random seed of 0.1234.



(a) Disturbance sensitivity versus input sensitivity.



(b) Disturbance sensitivity versus closed loop transfer function.



(c) Input sensitivity versus closed loop transfer function.

Figure 5.2: Pareto optimal fronts for different combinations of fitness functions for the simple control problem.

the limited space allowed for such a publication, there were many details and nuances which had to be omitted. Because of that, the issue will be reexamined here along with the omitted materials.

The reason why the scaling came to have such an effect on the result becomes apparent when taking a closer look at the crowding calculations. For the two-dimensional case, the crowding distance for individual j when sorted in ascending order of fitness function f_1 is calculated based on the two neighbors $j-1$ and $j+1$ given by

$$d_j = \begin{cases} \frac{f_{1,j+1}-f_{1,j-1}}{f_{1,max}-f_{1,min}} + \frac{f_{2,j+1}-f_{2,j-1}}{f_{2,max}-f_{2,min}} & 0 < j < N_{front} \\ \infty & \text{otherwise} \end{cases}, \tag{5.4}$$

where N_{front} is the number of individuals belonging to the current front. If one of the neighboring individuals does not exist $j = 0$ or $j = N_{front}$, making j an endpoint, then d_j is set equal to ∞ to ensure that the endpoints will be emphasized over the rest of the individuals of that front.

When comparing that crowding distance calculation to the case of the simple control problem described above, it becomes clear that the normalization terms $f_{1,max} - f_{1,min}$ and $f_{2,max} - f_{2,min}$ for the control problem cannot be determined prior to running the algorithm. As NSGA-II needs the fitness functions to be normalized in such a way prior to execution of the algorithm, no normalization could be performed for the simple control problem and the crowding distance calculations were thus reduced to

$$d_j = \begin{cases} \Delta f_{1,j} + \Delta f_{2,j} & 0 < j < N_{front} \\ \infty & \text{otherwise} \end{cases}, \tag{5.5}$$

where $\Delta f_{1,j} = f_{1,j+1} - f_{1,j-1}$ and $\Delta f_{2,j} = f_{2,j+1} - f_{2,j-1}$. If one of the fitness functions then turns out to contain larger values than another, e.g. $\Delta f_{1,j} \gg \Delta f_{2,j}$, then the crowding distance will become biased towards that fitness function $d_j \approx \Delta f_{1,j}$. In some cases, this approximation does not have a devastating effect on achieving an acceptable spread of solutions as seen in figure 5.3.

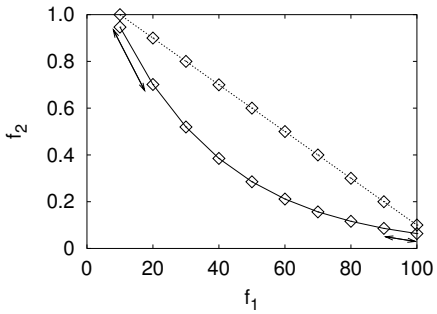


Figure 5.3: For fronts with small gradual slope changes an acceptable distribution can be obtained even if one of the fitness functions (in this case f_2) is neglected from the crowding distance calculations.

Even though the crowding distances marked by the arrows differ considerably in size, the spread

of solutions still cover the front relatively well. Unfortunately, not all combinations of fitness functions will result in fronts exhibiting such a property. An illustration of this is shown in figure 5.4 where a piecewise linear front with bad scaling will result in a poor distribution of solutions when one of the fitness functions is neglected from the crowding distance calculations.

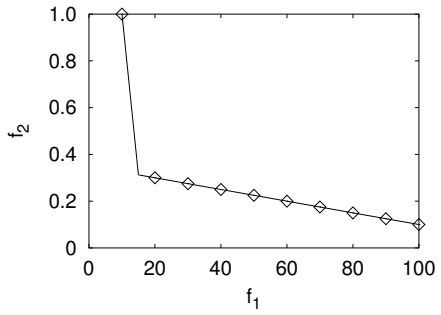


Figure 5.4: Neglecting one of the fitness functions in the crowding distance calculation on a piecewise linear front with bad scaling results in a bad distribution.

As a result, a large area of the front is ignored making it harder to make decisions between the different solutions because of their bias towards one of the fitness functions, namely the one with the largest values. This happens in spite of the fact that no preference between the fitness functions might have been made and is thus not desirable. Only if a fitness function is specifically indicated to be emphasized more than others should such a bias be present. In this chapter, the preferred case for a problem will be one where the solutions are uniformly distributed along the front no matter if the problem is badly scaled or not, thus requiring that all fitness functions must have an influence on the crowding distance. The reason why a uniform distribution is desired comes from the fact that any other desired distribution of solutions can be obtained by applying a corresponding mapping to the set of solutions. The uniform distribution can thus be used as a base for any other desired distribution of solutions. For the badly scaled case of figure 5.4, a corresponding uniform distribution is shown in figure 5.5 on the following page.

Having identified the problem that NSGA-II is not capable of finding a uniform distribution of solutions for an arbitrarily chosen problem, the time has now come to investigate the issue of uniform distributions further such that the problem can be corrected.

5.3 Uniform Distributions

When it comes to obtaining uniform distributions, the aim is to obtain an equal spacing between the different solutions in fitness space such that it is possible to make decisions based on a varied set of solutions. As it might have become clear from the observations mentioned previously, there are two different scenarios when it comes to the possible scaling of the different fitness functions such that uniform distributions can be obtained. Those two scenarios are those where the bounds of the fitness functions are either known or unknown, and each of these scenarios will be discussed

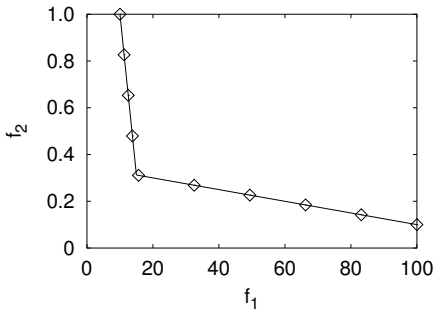


Figure 5.5: A good uniform distribution of solutions on a piecewise linear front with bad scaling.

in turn. First, let us take a look at the case with the known bounds.

Known Bounds For problems whose bounds are known in advance, there have been quite a lot of research into developing crowding mechanisms or other methods that would produce a uniform distribution of solutions along the Pareto optimal front. Some of this research can be found in Obayashi (1997) and Deb, Mohan, and Mishra (2003). As such, when the bounds of the individual fitness functions are known, this information can be used a priori to normalize the fitness functions, just as seen in the crowding calculations of NSGA-II, in equation (5.4). Unfortunately, this is very restrictive since it not only requires a certain amount of knowledge about the fitness functions prior to running the algorithm, but it also does not guarantee that the solutions will be uniformly distributed. In some of the literature, the values for $f_{i,min}$ and $f_{i,max}$ are recommended to be set equal to the minimum and maximum obtainable values of the individual fitness functions f_i (Deb, 2001). This is far from optimal, since the Pareto optimal solutions might only be located in part of the known range of the different fitness functions, thus resulting in the normalization being performed based on incorrect values. Also, in Bentley and Wakefield (1998), a dynamic approach for setting the values of $f_{i,min}$ and $f_{i,max}$ for calculation of fitness values was suggested, but since the work considered the ranking method separately from the other fitness calculation methods, it is not possible to extend the results directly to NSGA-II.

Unknown Bounds From the example given in section 5.1 on page 84 where it was not possible to use the normalization terms $f_{i,min}$ and $f_{i,max}$ because of the values being unbounded, it is clear that some sort of scheme must be used in order to normalize the fitness functions and avoid the situation of bad scaling. That is the only way to keep any fitness function from dominating another due to excessive fitness values. An attempt to use fixed values to normalize the fitness functions would not be optimal for the case of unknown bounds, since it might be very difficult to find the appropriate values, and obtaining an acceptable distribution of solutions might also be difficult to obtain due to the continually changes to the fitness values during the evolutionary process. It is thus necessary to use a method that changes the values dynamically in a way similar to the one suggested in Bentley and Wakefield (1998) or to use another crowding technique such as those

seen in PAES (Knowles & Corne, 1999) and PESA (Corne, Knowles, & Oates, 2000).

To accommodate for the case with unknown bounds, two methods for dealing with this issue will be discussed, of which one is similar to the method proposed in Bentley and Wakefield (1998).

The two normalization schemes presented here will be applied either globally or locally. When applied globally, the maximum and minimum values for each of the fitness functions will be found based on the entire population in each generation and used for normalization, which is similar to the approach used in Bentley and Wakefield (1998). For the locally based normalization scheme, the maximum and minimum values for each fitness function will be found amongst those individuals belonging to the same front in each generation. As such, for the locally based normalization scheme, the normalization will be different for each front. Using one of these methods should make it possible to obtain a good uniform distribution of solutions along the Pareto optimal front for situations where the bounds of the fitness functions are unknown. So, it is now time to discuss some of the details of the experiments, such that the normalization methods can be investigated.

5.4 Experimental Setup

The experiments will emphasize on the usability of the two proposed normalization methods. To do that with the highest degree of confidence, the experiments will be specifically designed for investigating this matter. So, let us first take a look at the fitness functions that will be used.

The number of fitness functions will remain at two such that it can be easily visualized. However, the results obtained will also be applicable for higher dimensions.

The fitness functions used for this investigation of the normalization methods are deliberately chosen such that bad scaling is easily obtained and such that the optimal Pareto front is known. It is thus easy to verify whether the solutions found actually belong to the Pareto set or not and the issue of investigating the distribution of the solutions can take priority in the processing of the results.

To emphasize on the bad scaling between the fitness functions, the fitness functions are chosen in such a way that the relative scaling between them can easily be varied. The fitness functions used are not based on existing benchmark problems for MOEAs such as those found in Knowles and Corne (1999), Deb (1999), and Zitzler and Thiele (1999) because these do not have any scaling issues. Also, the previously mentioned control problem is not used as a basis for the fitness functions because the Pareto optimal front for that problem is not known. Instead, an artificial set of fitness functions is constructed as given by

$$\min f_1(x) = x_1^k + |x_2|, \qquad k > 0 \tag{5.6}$$

$$\min f_2(x) = x_1^{-l} + |x_2|, \qquad x_1 > 0, \;\; l > 0 \;\; , \tag{5.7}$$

where $x_1, x_2 \in \mathbb{R}$ are the optimization variables, and k and l determine the extent of scaling between the functions. The optimal Pareto-front is obtained for

$$f_2 = f_1^{-\gamma} \qquad x_2 = 0 \;\; , \tag{5.8}$$

where $\gamma = \frac{l}{k}$. For small values of γ ($\gamma < 1$), the fitness function f_1 will have bigger scaling than f_2 and vice versa. An equal scaling between the fitness functions will be obtained for $\gamma = 1$.

With the fitness functions in place, it is just a matter of choosing the remainder of the parameters for the experiments. Because the NSGA-II algorithm is capable of using both binary and real value representations, the experiments will be performed for both of these. First, let us take a look at the parameters used for the binary case.

5.4.1 Binary Case

The parameters chosen for the binary case are summarized in table 5.3.

Parameter description	Type	Value	Designation
Number of fitness functions	Functions	2	f_1, f_2
Number of variables	Binary	2	x_1, x_2
x_1 variable	Binary 16 bit	$[0.1, 10]$	$[x_{1_{min}}, x_{1_{max}}]$
x_2 variable	Binary 16 bit	$[-100, 100]$	$[x_{2_{min}}, x_{2_{max}}]$
Selection operator	Tournament	2	s_{size}
Crossover operator	Uniform	0.5	-
Crossover probability	-	0.9	p_c
Mutation operator	Bitwise	-	-
Mutation probability	-	0.03125	p_m
Population size	-	200	N
Maximum generations	-	200	τ_{EA}

Table 5.3: The parameters used in NSGA-II for the investigation of uniform scaling issues using binary variables.

It is expected that the issue of scaling should not be dependent on the underlying representation but to be sure the representation of the variables is chosen to be based on both binary and real values.

For the binary case, the range of x_1 is chosen to belong to $[0.1, 10]$ such that it complies with the constraints laid forth in (5.7). Based on the fact that the two variables have a representation of 16 bits each, it is possible to determine that the search space is comprised of 2^{32} possible solutions, of which 2^{17} will be located on the Pareto optimal front. The reason why the number of solutions on the Pareto optimal front is found to be 2^{17} is because of the discretization done by the conversion into binary values where x_2 will have optimal values for both 0.001526 and -0.001526. It is thus clear that the problem is very simple and that the algorithm should have no difficulties in finding the Pareto optimal front.

When it comes to the selection and mutation operators, these are used with the default values of NSGA-II. This includes tournament selection with size 2 and bitwise mutation with a mutation probability of $\frac{1}{l_{total}}$, where l_{total} is the total length of the bit-string representing an individual. Because the problem is so simple, it should not matter what kind of crossover is used and as such the disruptive uniform crossover is chosen with a value of 0.5 which indicates the probability for each bit to be crossed for this crossover scheme. This probability is not to mistake for the crossover probability of 0.9 which is the probability that an individual will be selected for crossover. Further,

with a crossover probability of 0.9, it is expected that the algorithm will have a good convergence for the simple problem under consideration, since such a value should place the combination of values for selection pressure and crossover probability well into the "sweet spot" as given in Goldberg (2002).

For a population size of 200 it will be possible for the algorithm to cover 0.3% of the Pareto optimal front if all of the individuals are located on the front and non-overlapping. Thus, the coverage will be quite loose giving ample opportunity for the algorithm to explore the front using the crowding mechanism.

A maximum generation number of 200 is chosen to ensure that the algorithm will have more than enough time to converge on the Pareto optimal front and for the crowding mechanism to take effect. Even though it is expected that the algorithm will converge on the front quite early in the run, it is important that the crowding mechanism is given enough time to propagate properly. As such, by choosing the maximum generation number so high, should ensure that the crowding has been applied to the fullest extent, which should also be possible to see from the fact that the distribution of solutions should remain more or less constant over many of the last generations.

Because the uniformity of the distributions will be presented as the mean of the crowding distance, it is deemed necessary to perform several different runs such that the data will be valid from a statistical point of view. As such, the algorithm will be run with 30 different random seeds. When illustrating some of the distributions in the figures, it will, however, always be for a run using a random seed of 0.1234.

5.4.2 Real Valued Case

For the real valued case the parameters chosen are summarized in table 5.4.

Parameter description	Type	Value	Designation
Number of fitness functions	Functions	2	f_1, f_2
Number of variables	Reals	2	x_1, x_2
x_1 variable	Real	$[0.1, 10]$	$[x_{1_{min}}, x_{1_{max}}]$
x_2 variable	Real	$[-100, 100]$	$[x_{2_{min}}, x_{2_{max}}]$
Selection operator	Tournament	2	s_{size}
Crossover operator	SBX	10	-
Crossover probability	-	0.9	p_c
Mutation operator	Polynomial	50	-
Mutation probability	-	0.5	p_m
Population size	-	200	N
Maximum generations	-	200	τ_{EA}

Table 5.4: The parameters used in NSGA-II for the investigation of uniform scaling issues using real valued variables.

To be able to compare the results of the binary and real valued cases, the problem specific para-

meters are the same as those for the binary case. When it comes to the operator specific values, they are once again chosen such that they correspond to the default values of NSGA-II.

Now, with the setup in place it is now time to investigate the results obtained when running the algorithm.

5.5 Experimental Results

This section will discus the results obtained when running NSGA-II on the badly scaled fitness functions (5.6) and (5.7). For all of the experiments, the algorithm did not have any problems converging to the Pareto optimal front. Furthermore, the algorithm was capable of finding and preserving the outermost solution points of the Pareto optimal front for the problems as well. Because of these traits, the calculations that will be performed in this section will not include any penalizing terms caused by lack of convergence to the Pareto optimal front or lack of finding the outermost solutions. Now, let us take a look at how the original crowding measure of NSGA-II was able to handle the badly scaled fitness functions.

5.5.1 Original NSGA-II

To begin with, the algorithm was executed using 3 distinct γ-values (0.2, 1, and 5) with the original NSGA-II crowding mechanism and using binary representation. As such, it was possible to illustrate the effects of bad scaling for both of the cases where either f_1 or f_2 were overly emphasized. Similarly, the result of running the algorithm on the optimal situation, namely where the scaling on f_1 and f_2 were equal, could be found as well. The result of these 3 runs are shown in figures 5.6, 5.7a, and 5.7b along with the known Pareto optimal front.

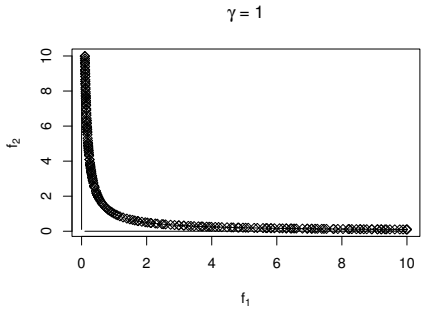


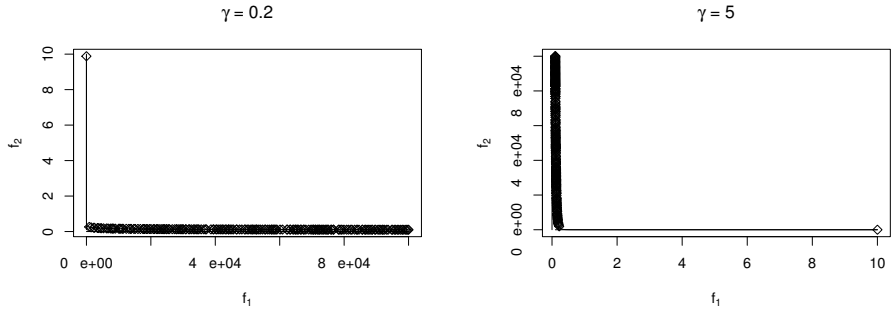
Figure 5.6: Pareto non-dominated front for $\gamma = 1$ using original NSGA-II crowding with binary representation and plotted on the known optimal front.

In figure 5.6, it is clear that when the scaling of the fitness functions are equal $\gamma = 1$ then NSGA-II is capable of not only finding the Pareto optimal front, but also to maintain a near uniform

distribution of solutions along the front. In order to put figures on the level of uniformity, it is possible to calculate the spread of the distribution for each front in a single run as

$$s_{dist} = \sum_{j=1}^{N_{front}-1} \left(\hat{d}_j - \bar{d}\right)^2 \quad , \tag{5.9}$$

where $\hat{d}_j$ is the normalized Euclidean distance between neighboring solutions and $\bar{d}$ is the mean of the normalized Euclidean distance between solutions. The spread calculation is thus a modified version of the one used in Deb (2001) with the main difference that the values $\hat{d}_j$ and $\bar{d}$ are normalized according to $f_{i,max} - f_{i,min}$ for each fitness function before calculation of the Euclidean distance and thus also before being included in the calculation of the spread. The calculation of the spread does not include the distance between the outermost solutions of the Pareto front and the known extremes of the Pareto front because the algorithm was indeed capable of finding these outermost solutions of the Pareto front. As such, the inclusion of such a term would simply correspond to an addition of a zero term. Also, since it is the spread of the non-dominated front which is of interest the calculated spreads will only be based on that particular front. Averaging the spread over 30 independent runs, the result of running NSGA-II using the original crowding measure yields a mean spread of $5.275 \cdot 10^{-3}$ for $\gamma = 1$. Having seen the result for the case of equal scaling between fitness functions with $\gamma = 1$, it is now time to see what happens for the badly scaled cases of $\gamma = 0.2$ and $\gamma = 5$. This is shown in figure 5.7.



(a) Small γ value.

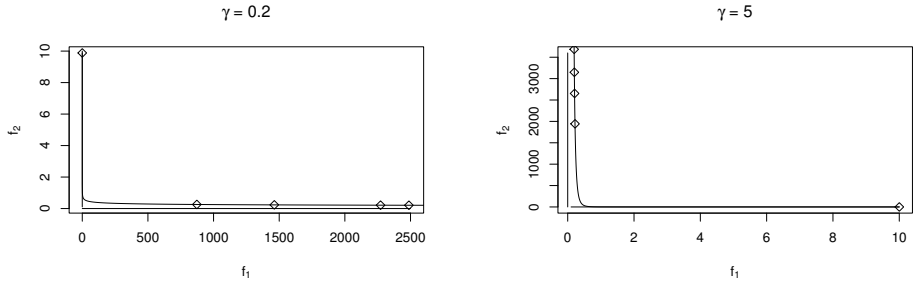
(b) Large γ value.

Figure 5.7: Pareto non-dominated fronts for large and small γ values using original NSGA-II crowding with binary representation and plotted on the known optimal front.

It can be seen clearly from the figure that even though the Pareto optimal front is found, the distribution of points for both $\gamma = 0.2$ and $\gamma = 5$ is not uniformly distributed. As such, from figure 5.7a it is seen that the tradeoffs for fitness function f_2 is practically non-existent because all of the solutions are biased towards fitness function f_1. When it comes to the calculated spread, the value obtained for $\gamma = 0.2$ is $955.1 \cdot 10^{-3}$, which is much higher than the spread obtained for

$\gamma = 1$. Similarly, for the opposite case in figure 5.7b where $\gamma = 5$, the solution points are biased towards fitness function f_2 and the calculated spread is found to be $964.2 \cdot 10^{-3}$.

It is quite clear, it is due to the fitness functions not being normalized that such poor distributions of points for the two extreme cases, where $\gamma = 0.2$ and $\gamma = 5$, are obtained. The dominant fitness function is thus too dominant in the crowding calculations and results in a poor distribution. A more detailed view of the effect the emphasized fitness functions from figure 5.7 on the preceding page have can be seen in the enlarged versions shown in figure 5.8.



(a) Small γ value with f_1 values less than 2500.

(b) Large γ value with f_2 values less than 3600.

Figure 5.8: Lower region of the Pareto non-dominated front for small and large γ values using original NSGA-II crowding and binary representation.

In those figures, it is quite clear how the fitness function with the largest scale totally dominate the crowding distance calculations.

As previously mentioned, a number of experiments were also performed using real valued representation. For those, the general conclusion is the same since the calculated spreads are $945.2 \cdot 10^{-3}$, $3.483 \cdot 10^{-3}$, and $924.9 \cdot 10^{-3}$ for corresponding γ values of 0.2, 1, and 5. One interesting thing to notice from the experiments using real valued encoding was that for a γ value of 5, the algorithm on some occasions had trouble finding the true Pareto optimal front and also on other occasions was not capable of finding one of the extremes of the Pareto optimal front. The failure to converge to the true Pareto front happened because the variable x_2 converged on small values that were not equal to the optimal value of 0. An illustration of one of the cases where the algorithm failed in finding the outermost solutions of the Pareto front is given in figure 5.9 on the facing page.

However, even though the algorithm was incapable of finding the extreme solutions in some cases and the fact that the calculation of the spreads did not punish this flaw, the calculated spread for the case of $\gamma = 5$ was only marginally better than the rest of the results for badly scaled cases and still far from uniform.

Thus, it should now be clear how a badly scaled problem with unknown bounds can result in a poor distribution of solutions along the Pareto optimal front for NSGA-II. In the cases shown previously, the bias towards one of the fitness functions for badly scaled problem have practically

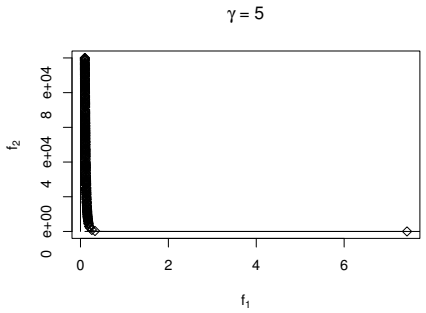


Figure 5.9: Pareto non-dominated front for $\gamma = 5$ using original NSGA-II crowding with real valued representation and plotted on the known optimal front.

eliminated any real tradeoff since all of the solutions are biased towards that particular fitness function. With this issue in mind, we can then proceed to discuss one of the proposed methods for coping with the scaling issue, namely global scaling.

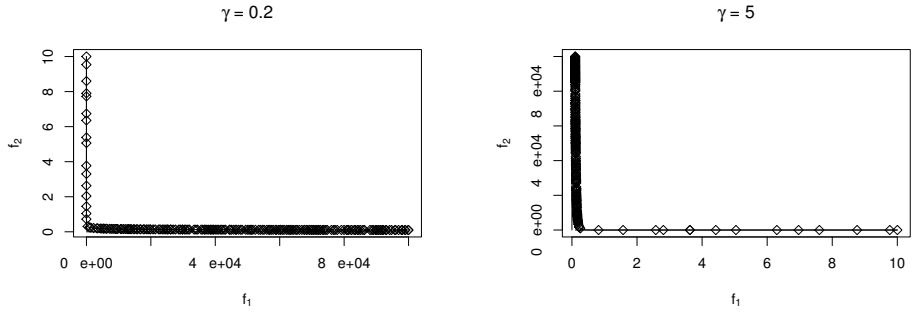
5.5.2 Global Scaling

First, it will be tested how the method of global scaling will affect the distribution of solutions along the Pareto optimal front for the badly scaled fitness functions of (5.6) and (5.7). As previously mentioned, the method of global scaling will be used to normalize the crowding distance calculations based on the maximum and minimum values of the different fitness functions present in the population at each generation. Thus, the method is quite similar to the one proposed in Bentley and Wakefield (1998). The results obtained for this normalization scheme using binary encoding are shown in figure 5.10 on the next page.

As seen in the figures, the distribution of points for the binary representation have improved over that of the original NSGA-II crowding, since there are now more than just one solution located on the Pareto optimal front near the fitness function with the smallest values. However, it is also clear that the distribution of solutions is still not uniformly distributed, meaning that the normalization scheme helped, but just not enough. This can also be concluded from the calculated spreads, which are $61.55 \cdot 10^{-3}$ for $\gamma = 0.2$ and $74.84 \cdot 10^{-3}$ for $\gamma = 5$.

For comparison, a test of the problem was also performed using $\gamma = 1$ which resulted in a spread of $6.238 \cdot 10^{-3}$. The spread for $\gamma = 1$ is a bit higher than that obtained using the original NSGA-II crowding, but the difference is not significant when compared to the poor distribution of points for the badly scaled cases.

Using a real valued representation, it was possible to obtain corresponding spreads of $11.02 \cdot 10^{-3}$, $6.200 \cdot 10^{-3}$, and $8.148 \cdot 10^{-3}$ for γ values of 0.2, 1, and 5 respectively. For the case of $\gamma = 0.2$ the algorithm once again had a little problem converging to the true Pareto optimal front for all experiments. However, it is clear from the spreads that the distribution of points for the badly scaled cases are almost as good as for the case with equal scaling between fitness functions. In



(a) Small γ value.

(b) Large γ value.

Figure 5.10: Pareto non-dominated fronts for small and large γ values when using globally based normalization and binary representation.

figure 5.11 on the facing page, one of the experiments using both a small and a large γ value and a random seed of 0.1234 are shown and it is clear that the distribution of points are practically uniform.

While the results of the first method using a globally based scaling method is now clear, it is time to take a look at the other proposed method of using a locally based scheme for normalization.

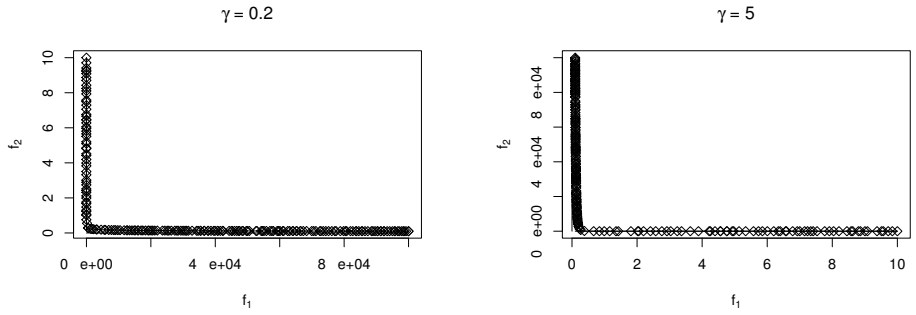
5.5.3 Local Scaling

The main idea of the locally based normalization scheme was to find the values used for normalization frontwise. This means that for each front the maximum and minimum values for each of the fitness functions are found and used for normalization. Running the algorithm using the locally based normalization scheme resulted in figure 5.12 on the facing page when using the binary representation of the individuals.

The figure shows that the distribution of points for both of the badly scaled problems are nearly uniform. This is also confirmed by the calculated spreads which are found to be $8.046 \cdot 10^{-3}$, $5.093 \cdot 10^{-3}$, and $7.176 \cdot 10^{-3}$ for corresponding γ values of 0.2, 1, and 5. So, even though the case for $\gamma = 1$ is still slightly better when looking at the calculated spreads, it is evident that the difference is minute and that the badly scaled problems thus have uniform distributions similar to those obtained for the problem with the equally scaled fitness functions.

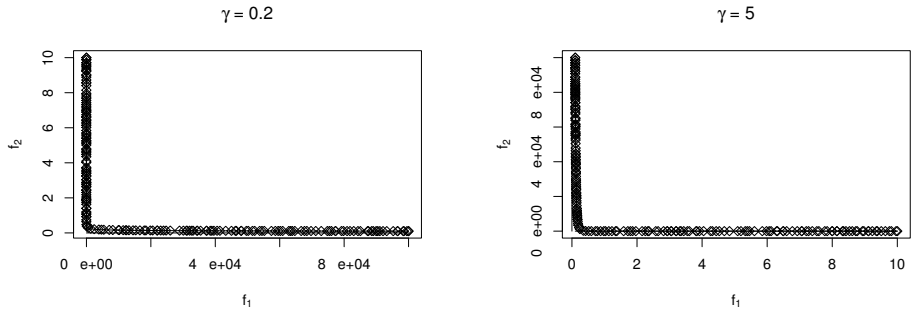
Similar results emerge when looking at the experiments involving the real valued representation. The spreads obtained are $4.756 \cdot 10^{-3}$, $4.948 \cdot 10^{-3}$, and $6.286 \cdot 10^{-3}$ for γ values of 0.2, 1, and 5. Also, the results for the real valued case can be seen in figure 5.13 on page 100.

Using the locally based normalization scheme, it was thus possible to obtain a near uniform distribution of solutions for the two badly scaled problems using both binary and real valued represen-



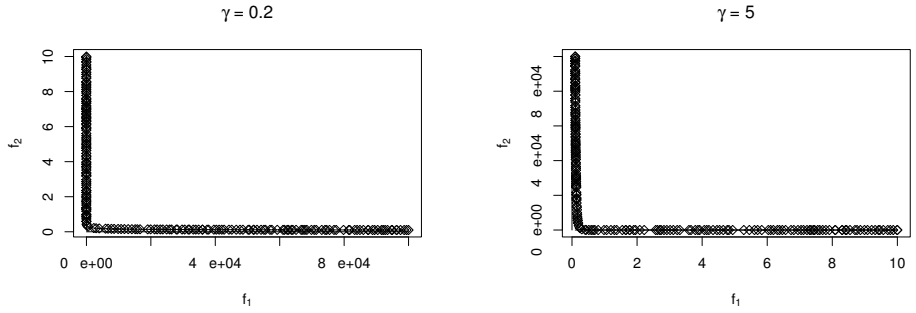
(a) Small γ value. (b) Large γ value.

Figure 5.11: Pareto non-dominated fronts for small and large γ values when using globally based normalization and real valued representation.



(a) Small γ value. (b) Large γ value.

Figure 5.12: Pareto non-dominated front for small and large γ values when using locally based normalization and binary representation.



(a) Small γ value.

(b) Large γ value.

Figure 5.13: Pareto non-dominated front for small and large γ values when using locally based normalization and real valued representation.

tations. Before moving on to the discussion of the results, there is one more issue that is relevant, and that issue is the effect of bias.

5.5.4 Introducing Bias

To show the effect of using a bias, the algorithm was also run using a γ value of 5 with locally based normalization and a biasing factor of 5 in favor of f_1. The biasing factor is implemented such that the crowding distance for f_1 in (5.4) is multiplied with 5 whereas the crowding distance for f_2 is not multiplied with anything, which will result in the desired bias. The experiment is performed using a binary representation and the result is shown in figure 5.14.

It is clear that when a bias is introduced, the density of solution points near the biased fitness function is increased, which is further emphasized by a calculated spread of $25.70 \cdot 10^{-3}$. Biasing can thus be used to control the density of the solutions near each fitness function. This phenomenon has already been investigated somewhat in other publications, such as Deb (2001). Now with the issue of biasing in place, it is now possible to come with some conclusions on the obtained results.

5.6 Concluding Remarks

The results obtained throughout this chapter are summarized in table 5.5 on the facing page.

From the table it is clear that for the binary representation the global scaling method was capable of partially solving the problem of poor distributions and that it required the use of the locally based normalization scheme to fully resolve the issue.

As for the real valued representation, the global scaling method was quite successful at obtaining

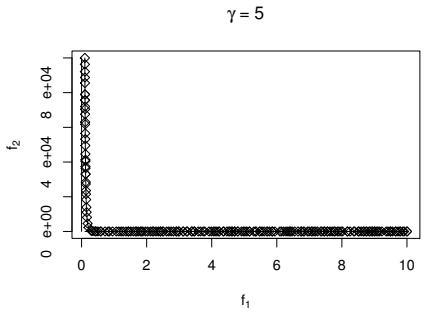


Figure 5.14: Pareto non-dominated front for large γ value using a biased normalization of fitness functions and binary representation.

Crowding method	Representation	$\gamma = 0.2$	$\gamma = 1$	$\gamma = 5$
Original	Binary	$955.1 \cdot 10^{-3}$	$5.275 \cdot 10^{-3}$	$964.2 \cdot 10^{-3}$
Global	Binary	$61.55 \cdot 10^{-3}$	$6.238 \cdot 10^{-3}$	$74.84 \cdot 10^{-3}$
Local	Binary	$8.046 \cdot 10^{-3}$	$5.093 \cdot 10^{-3}$	$7.176 \cdot 10^{-3}$
Biased	Binary	-	-	$25.70 \cdot 10^{-3}$
Original	Reals	$945.2 \cdot 10^{-3}$	$3.483 \cdot 10^{-3}$	$924.9 \cdot 10^{-3}$
Global	Reals	$11.02 \cdot 10^{-3}$	$6.200 \cdot 10^{-3}$	$8.148 \cdot 10^{-3}$
Local	Reals	$4.756 \cdot 10^{-3}$	$4.948 \cdot 10^{-3}$	$6.286 \cdot 10^{-3}$

Table 5.5: Spreads obtained using different γ values for original NSGA-II crowding distance calculations and for globally and locally based normalization of crowding distances.

a near uniform distribution, even though the spreads for the badly scaled problems ($\gamma = 0.2$ and $\gamma = 5$) are slightly larger than for the case of $\gamma = 1$.

The scheme that performed the best was the one based on local scaling, and that is because it used only the values present in each front for the normalization. Thus, if the population had converged on the Pareto optimal front and had found the outermost solutions, then the normalization would scale the fitness functions such that they in the crowding calculations would contribute equally.

The explanation for why the global normalization scheme only partially resolved the problem of the non-uniform distribution of solutions on the Pareto front for the badly scaled problem using binary representation lies in fact that the Pareto front did not cover the entire space of feasible solutions. Whereas the fitness values of f_1 for the case of $\gamma = 0.2$ could lie in the interval $[0, 10100]$ and $[0, 110]$ for f_2, the Pareto optimal front only extended as far as $[0, 10000]$ for f_1 and $[0, 10]$ for f_2. Because the binary representation used uniform crossover and bit-flip mutation, it was highly possible for an individual located on the Pareto optimal front to create an offspring

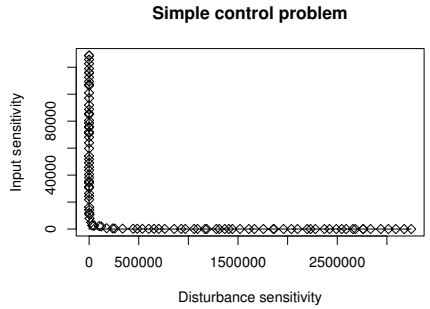
that would be located far from itself. Using the global scaling method, it was then possible for a population converged on the Pareto optimal front to create offspring with a fitness value near 110 for f_2 which then would be used to scale the solutions in the interval $[0, 10]$ on the Pareto optimal front. As such, the normalization of f_2 could be using a scaling of $\frac{10}{110} = \frac{1}{11}$. This actually corresponds to the effect seen by the introduction of the biasing factor, and thus explains why the distribution of solutions improved only partially. A poor individual that was not part of the Pareto optimal set could thus cause a biasing effect on the Pareto optimal solutions.

However, because the real valued representation used crossover and mutation operator that emphasize on the offspring being located near the parent with high probability, it was not as likely for the real valued representation to create offspring with very lousy fitness values. As such, once the Pareto optimal front had been found, the offspring created using one of these operators would most likely be located in the vicinity of the Pareto optimal front and the normalization would thus be close to a normalization based on local scaling.

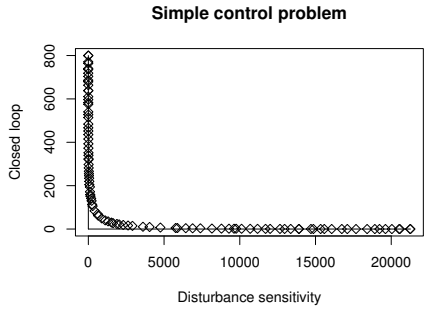
In this chapter, the issue of poor distribution of solutions on the Pareto front for badly scaled fitness functions has been investigated. The issue has also been successfully resolved using a local normalization scheme based on the maximum and minimum values of each fitness function for each of the fronts. It cannot be guaranteed that the proposed method will be successful for any problem encountered, but the empirical testing performed here on very badly scaled problems indicate that the method should work for other cases as well.

The issue was first identified while performing tests of NSGA-II on a simple control problem, and as can be seen from figure 5.15 on the facing page, the locally based normalization method is successful at providing a nearly uniform distribution of possible tradeoff solutions for that particular problem.

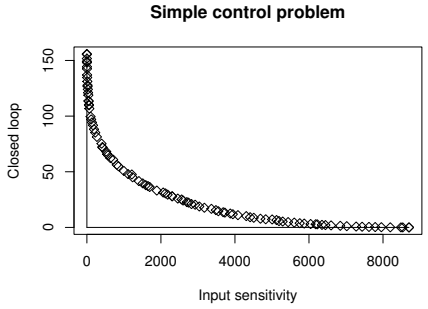
As will become apparent in the next chapter, it is not always desired to have a uniform distribution of solutions on the Pareto optimal front. However, if a problem is badly scaled, it is necessary to first use a normalization scheme in order to be able to employ other methods before the distribution of points can be manipulated in the desired way. In the next chapter it will be discussed how to obtain different biased distributions that can emphasize on certain aspects of a problem. However, because of the findings of this chapter the applications of NSGA-II to all problems in the remainder of this thesis will use the locally based normalization scheme unless otherwise stated. This ensures that any issues involving disparate scaling of fitness functions will not have an effect on the results obtained.



(a) Disturbance sensitivity versus input sensitivity.



(b) Disturbance sensitivity versus closed loop transfer function.



(c) Input sensitivity versus closed loop transfer function.

Figure 5.15: Pareto optimal fronts for different combinations of fitness functions for the simple control problem using locally based normalization.

Chapter 6

Emphasizing Curvature using Crowding

In the previous chapter, it became clear how the issue of disparate scaling of fitness functions could result in a non-uniform distribution of solution points along the Pareto optimal front. Two different methods were proposed for resolving the issue and one of those methods did successfully resolve it. This means that despite the disparate scaling of the fitness functions, it was possible to obtain a uniform distribution of points along the Pareto optimal front. Such a uniform distribution is, however, not always desirable. How to create distributions of points that are non-uniform, but which emphasize on relevant areas of the Pareto optimal front, will be the focus of this chapter along with several different ways of achieving this goal. The methods proposed will focus on estimations of derivatives and the curvature that can be obtained for pairwise combinations of fitness functions for a given problem. However, let us first examine the need for non-uniform distributions.

6.1 Non-Uniform Distributions

When it comes to the tradeoffs along the Pareto optimal front, it might not always be desirable to obtain a uniform distribution of points. An example of this is illustrated in figure 6.1 on the next page.

In the figure, the 5 white points indicate a uniform distribution of points on a possible Pareto optimal set. It is quite clear that in order to obtain a distribution which could give a better set of tradeoffs solutions, it would be preferable to move two of the points to the position indicated by the black points as illustrated in the figure. By doing so, the points would give a better image of the tradeoffs that is apparent from the illustrated front. However, for the example, it is also apparent that it does not give an optimal representation of the tradeoffs. An optimal representation of the tradeoffs would be obtained if it was possible to somehow emphasize the curvature of the front.

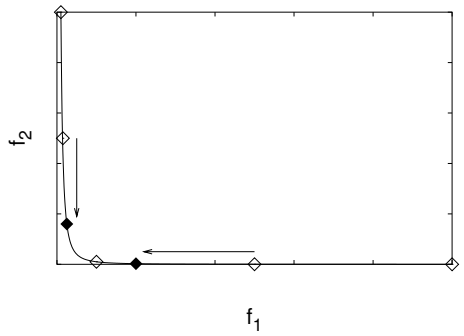


Figure 6.1: A more desirable non-uniform distribution of solution points can be obtained by moving some of the uniformly distributed solution points.

This fact is what this chapter is based on. Several methods will be investigated as to how well they are able to emphasize on the curvature. Some investigation into this field has been done earlier in Branke, Deb, Dierolf, and Osswald (2004). In that paper, two methods were proposed to find "knees" for multiple fitness problems. These knees are the points on the Pareto optimal front where a minor degradation in one fitness function would result in a major degradation of another, thus, indicating where the best tradeoffs can be obtained. An example of such a knee is shown in figure 6.2.

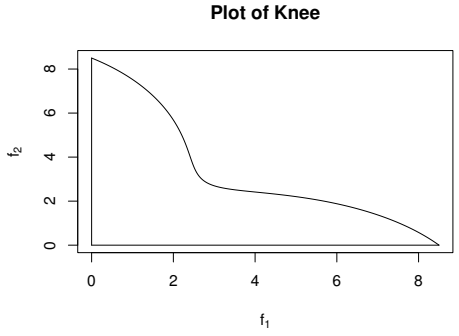
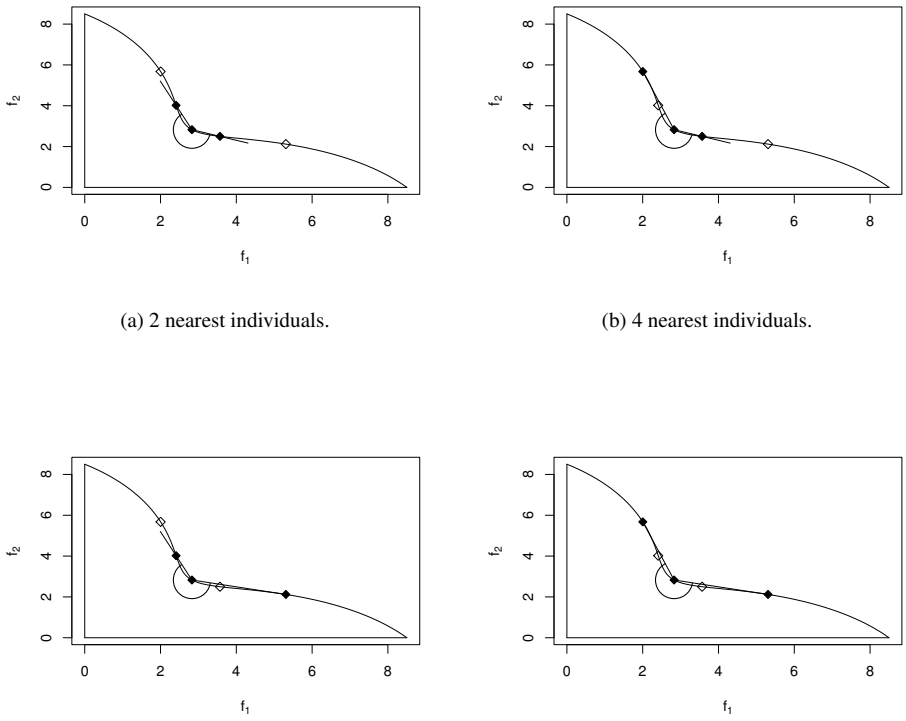


Figure 6.2: An illustration of a knee on a Pareto optimal front where a small change in one fitness function near the knee will result in a large change of the other fitness function.

One of the methods proposed was different ways to maximize the angle between neighboring individuals based on 2 or 4 individuals as shown in figure 6.3 on the facing page.

By choosing the maximum of the 4 different angles, it was possible for the authors to use this



(a) 2 nearest individuals.

(b) 4 nearest individuals.

(c) 4 nearest individuals.

(d) 4 nearest individuals.

Figure 6.3: Calculation of the angles between different individuals using a combination of (a) 2 individuals or (b, c, d) 4 individuals.

angle measure as the crowding criteria such that the individuals having the biggest angles, and thus also the largest tradeoffs, would be preferred.

The other method proposed was the use of expected marginal utility functions for linear utility functions of the form

$$U(x, \gamma) = \gamma f_1(x) + (1 - \gamma) f_2(x) \tag{6.1}$$

with all $\gamma \in [0, 1]$ being equally likely. If the user had a preference for such a utility function $U(x, \gamma')$ with a known γ', it was then possible to calculate the expected utility for the different individuals and make a preference of those having the highest expected utility. Because of the requirement of the linear utility functions, the second method could only find knees in concave regions of the Pareto optimal front and as such both of the methods were only tested in that respect, for finding knees on concave regions of the Pareto optimal front.

As the methods proposed in Branke, Deb, Dierolf, and Osswald (2004) only could find the knees for concave regions of the Pareto optimal front, this limited the tradeoffs to those regions only. For the cases where it might be desired to obtain an overview of the different tradeoffs of the entire Pareto optimal front, no investigation was performed.

The methods proposed in this thesis will focus on finding the tradeoffs for all regions of the Pareto optimal front including convex, concave and perhaps even linear regions. In this chapter, a total of 4 methods will be investigated with regard to emphasizing the tradeoffs regions. One will be based on estimates of the first derivative of neighboring individuals and an extension to that will be based on the formula for curvature, using estimates of the first and second order derivatives. The third method will focus on maximizing the angle between neighboring individuals, but in a different manner than the one proposed in Branke, Deb, Dierolf, and Osswald (2004). Finally, a method using estimates of the circumradius will be investigated. Because these methods are focused on using either angles or derivatives, it will only be possible to apply the method pairwise to two fitness functions at a time. The examples shown will only consider the two fitness case, but if is desired to use more than two fitness functions it would be necessary to perform the crowding calculation for each of the possible pairings of the fitness functions requiring $\frac{M(M-1)}{2}$ applications of the crowding scheme. Having for instance 3 fitness functions ($M = 3$) would require 3 applications of the method for (f_1, f_2), (f_1, f_3), and (f_2, f_3) respectively. However, doing so would not be recommended since some of the dominance properties would not hold true for the random pairing of two fitness functions for a problem of dimension three or higher.

In the description of all of the proposed methods, any duplicate individuals will not be considered in the calculations. This means that if more than one individual are located at the same point in space, the neighbors for those individuals will be considered as those individuals which are closest to that point on either side. This ensures that each individual will receive a valid crowding measure based on the estimated curvature or slope of the Pareto optimal front at the point where it is located. Now, with those details in place, let us first take a look at one of the derivative based methods and how it can be applied.

6.1.1 Derivative Based Crowding

The derivative based crowding scheme is based on the estimated first derivatives, the slopes, of the Pareto optimal front at the point of each of the individuals. Since the method is only considered for two fitness functions at a time, it is possible to calculate such an estimated first derivative for

each individual. This is done by using the two neighboring individuals to estimate the derivative as illustrated in figure 6.4.

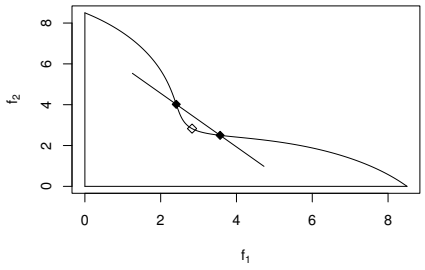


Figure 6.4: Estimating the first derivative of an individual using the two neighboring individuals.

The way the first derivative is calculated for an individual j can then be written as

$$f'_j = \frac{df_{1_j}}{df_{2_j}} = \begin{cases} \frac{f_{2_{j-1}} - f_{2_{j+1}}}{f_{1_{j-1}} - f_{1_{j+1}}} & 0 < j < N_{front} \\ \frac{f_{2_j} - f_{2_{j+1}}}{f_{1_j} - f_{1_{j+1}}} & j = 0 \\ \frac{f_{2_{j-1}} - f_{2_j}}{f_{1_{j-1}} - f_{1_j}} & j = N_{front} \end{cases} . \tag{6.2}$$

For the extreme cases, e.g. when $j = 0$ or $j = N_{front}$, the derivative is found using $j = 0$ and $j = 1$ or $j = N_{front} - 1$ and $j = N_{front}$ respectively such that the estimate does not depend on any non-existing neighbor.

One interesting fact is that the cases where the tradeoff in both objectives will be the largest are when the value of the first derivative is -1. Using this fact, makes it possible to formulate a measure which will attempt to emphasize on that specific case. This has lead to the first of the proposed measures that will emphasize on curvature using derivatives. The measure is given by finding the maximum value of

$$d_j = min(|f'_j|, |\frac{1}{f'_j}|) \ , \tag{6.3}$$

and will have a maximum value of $d_j = 1$ when $f'_j = 1$. Having described this first method of how emphasis can be put on the regions which have the most tradeoff in all objectives, the time has come to describe the second method which uses both first and second order derivative in an attempt to estimate the curvature of the Pareto optimal front at each individual.

6.1.2 Curvature Based Crowding

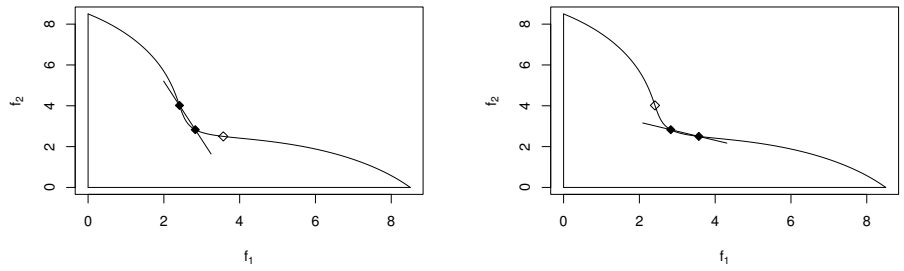
The method of curvature based crowding is based on the well-known formula of curvature given by

$$d_j = \frac{|f_j''|}{[1 + (f_j')^2]^{3/2}} \quad . \tag{6.4}$$

Since the formula for curvature contains both first and second order derivatives, these must be found using estimates. The first order derivatives can be found in the same manner as was used for the derivative based crowding, namely using the neighbors to estimate the first order derivative for the point of each individual. The curvature formula, however, also requires the use of a second order derivative. The way to find this is given by the formula

$$f_j'' = \frac{df_j'}{df_{1_j}} \begin{cases} \frac{f_{l_j}' - f_{r_j}'}{\frac{1}{2}(f_{1_{j-1}} - f_{1_{j+1}})} & 0 < j < N_{front} \\ \frac{-\frac{100}{\epsilon} - f_{r_j}'}{\frac{1}{2}(f_{1_j} - f_{1_{j+1}})} & j = 0 \\ \frac{f_{l_j}' - (-\frac{\epsilon}{100})}{\frac{1}{2}(f_{1_{j-1}} - f_{1_j})} & j = N_{front} \end{cases} , \tag{6.5}$$

where the first order derivative f_{l_j}' is found using $j-1$ and j and the other first order derivative f_{r_j}' is found using j and $j+1$. The way the first order derivatives f_{l_j}' and f_{r_j}' are found is illustrated in figure 6.5.



(a) f_{l_j}'. (b) f_{r_j}'.

Figure 6.5: The estimate for the second order derivatives are found using the estimated first order derivatives on each side of an individual.

The reason why formula (6.5) only uses half of the distance $f_{1_{j-1}} - f_{1_{j+1}}$ is because the corresponding estimated first order derivatives f_{l_j}' and f_{r_j}' are set to be located at $f_{1_{j-1}} + \frac{f_{1_j} - f_{1_{j-1}}}{2}$ and $f_{1_j} + \frac{f_{1_{j+1}} - f_{1_j}}{2}$ respectively, which correspondingly is halfway between $j-1$ and j and j

and $j + 1$. The only remaining issue to be discussed for the curvature based crowding measure is how the endpoints are treated. Since there is no neighbor to one side, both the first order derivative and the second order derivative cannot be estimated. So, when an endpoint is encountered, the estimate for the first derivative f'_j is calculated just as for the derivative based crowding measure from section 6.1.1 on page 108, whereas the missing first derivative estimate f'_{l_j} or f'_{r_j} is replaced by a value of $-\frac{100}{\epsilon}$ and $-\frac{\epsilon}{100}$ respectively, where epsilon is a small value provided by the user for handling that specific situation. It is thus possible when specifying ϵ to indicate how much the endpoints should be emphasized.

This concludes the description of the proposed crowding method using curvature. The next crowding measure, which emphasizes tradeoffs regions on the Pareto optimal front, to be discussed is the one based on angles.

6.1.3 Angle Based Crowding

The angle based crowding method which will be presented here is similar to the angle based method of Branke, Deb, Dierolf, and Osswald (2004). There are, however, two differences. First of all, it is only the nearest neighbor to each side which will be used for the angle calculation, as can be seen in figure 6.3a. The second difference is that where the method of Branke, Deb, Dierolf, and Osswald (2004) was only able to emphasize on knees in the concave regions of the Pareto optimal front the method proposed here should also be able to find the "ankles" of the convex regions as well. Such a situation is shown in figure 6.6 along with the angle spanned by 3 individuals near the ankle.

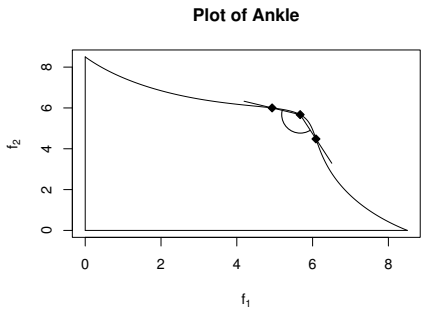


Figure 6.6: An illustration of an ankle on a Pareto optimal front along with the angle spanned by 3 individuals for that particular situation.

It is desired to both maximize the angle at 270° to find possible knees and to minimize the angle at 90° for finding ankles. Because of the dominance relation used when deciding on which individuals belong to the different ranks, it is not possible to obtain angles that are below 90° or higher than 270° since either the middle individual would be dominated by one of the side individuals or one of the individuals to the sides would be dominated by the middle individual respectively. Because of this desire to both minimize and maximize the angles, it was necessary to find a way

in which to do both. For this purpose the cosine of the angles was found to be useful. For angles in the range 90° to 270°, the cosine will have values ranging from -1 for 180° to 0 for both 90° and 270°. Thus, by taking the maximum of the cosine to the angles, it is possible to emphasize on the sharp angles 90° and 270° while a non-curved region having an angle of 180° would actually receive the lowest crowding value.

In order to use only positive values for the crowding measure, the angle based crowding measure is then formulated as

$$d_j = \begin{cases} 1 + \cos(\alpha_j) & 0 < j < N_{front} \\ 1 & \text{otherwise} \end{cases}, \tag{6.6}$$

where α_j is the angle found for individual j using its nearest neighbor to each side. For the case when an individual is located at an endpoint and thus does not have a neighbor to one side, a value of 1 is used such that the endpoints will also be emphasized. With this angle based method in place, the time has come to present that last of the proposed methods for emphasizing tradeoff regions of the Pareto optimal front.

6.1.4 Inverse Circumradius Based Crowding

The last method that will be presented, which should be able to emphasize on the different tradeoff regions of a Pareto optimal front is based on a geometrical principle applied to triangles, namely the enscribing circle of a triangle known as the circumradius. An illustration of the circumradius of a triangle is shown in figure 6.7.

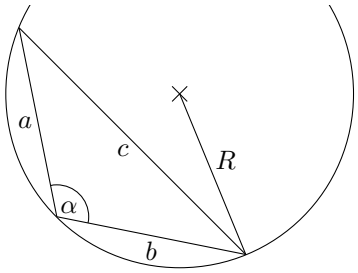


Figure 6.7: The circumradius of a triangle.

It can easily be seen how three neighboring individuals can be considered as the endpoints of the triangle and the circumradius for that triangle is then a measure of the curvature spanned by those individuals. The bigger the circumradius, the less curvature will be spanned by the individuals and as such it is the inverse of the circumradius which is used as a crowding measure since a large inverse would mean that the individuals span a highly curved area. The illustration in figure 6.7 is based on a knee, but it is also valid for an ankle, since the three individuals will always span a triangle. Interestingly, the triangle that is spanned by the individuals will always be obtuse, since any acute triangle would require one of the individuals to dominate another similar to the case of the angle based crowding measure.

Since the circumradius R can be expressed as

$$R = \frac{c}{2\sin(\alpha)} \quad 0 < \alpha < \pi \ , \tag{6.7}$$

the inverse circumradius will then be given by

$$R^{-1} = \frac{2\sin(\alpha)}{c} \quad 0 < \alpha < \pi \ . \tag{6.8}$$

The angle α will always lie in the specified interval since a value outside the range will correspond to a change of orientation of the triangle and switching from using the inner angle spanned by the two intersecting lines to using the outer angle or vice versa.

The crowding measure for an individual j can now be expressed as

$$d_j = \begin{cases} \frac{2\sin(\alpha_j)}{c_j} & 0 < j < N \\ \frac{100}{\epsilon} & \text{otherwise} \end{cases} \ , \tag{6.9}$$

where α_j and c_j are the corresponding α and c values when individual j is located at the intersection of the sides a and b.

The special case of the crowding measure is obtained for the endpoints where it is not possible to specify neither an angle α_j nor a length c_j. For this case, a value of $\frac{100}{\epsilon}$ is assigned such that the user can specify to what degree the endpoints should be emphasized.

Where the first two methods were based on derivatives, the last two of the proposed methods were based on geometrical principles. Because of the geometrical principles, it could be interesting to know exactly how the latter two methods differ from each other. The angle based method focused on maximizing $\cos(\alpha_j)$ for $90^\circ \leq \alpha_j \leq 270^\circ$. Since all of the values are in the range -1 to 0, the exact same result would actually be obtained if one were to minimize $\cos^2(\alpha_j)$ having values in the range 0 to 1. This can then be considered equivalent to maximizing $1 - \cos^2(\alpha_j)$, which due to the Pythagorean identity is identical to maximizing $\sin^2(\alpha_j)$.

For the circle based crowding it is the inverse of the circumradius which is sought maximized. As seen from equation (6.8), a maximization of the inverse of the circumradius corresponds to a maximization of $c^{-1}|\sin(\alpha_j)|$ for any angle $0^\circ \leq \alpha_j \leq 360^\circ$. This corresponds to a maximization of either of the terms $|\sin(\alpha_j)|$ or c^{-1}. The first of these terms is equivalent to a maximization of $\sin^2(\alpha_j)$, which has been shown to be equivalent to the angle based method. However, because of the latter term c^{-1}, the method of maximizing the inverse of the circumradius is not equivalent to the angle based method. In the attempt to maximize the inverse circumradius it is also attempted to minimize the length of c, which is the distance between the nearest neighbor to each side of individual j for which the crowding calculation is performed. The method thus not only attempts to find the maximum or minimum of the angle α_j, but also tries to keep the neighbors $j - 1$ and $j + 1$ as close together as possible.

Having proposed 4 different crowding methods for emphasizing different tradeoff regions of the Pareto optimal front, the time has now come to experimentally test the performance of each. First, we will take a look at the experimental setup that will be used in the tests.

6.2 Experimental Setup

Before the experiments can be performed, it is necessary to formulate the fitness functions that will be used for the experiments. Once these fitness functions have been formulated, the specific setup for each of the different experiments will be presented followed by the results of the experiments.

For the experiments it has been decided to use 3 of the same scenarios used in Branke, Deb, Dierolf, and Osswald (2004). The first two are based on the DO2DK problem given by

$$\begin{aligned}
\min f_1(x) &= g(x)r(x_1)\left(\sin\left(\pi x_1/2^{S+1} + \left(1 + \frac{2^S - 1}{2^{S+2}}\right)\pi\right) + 1\right) \\
\min f_2(x) &= g(x)r(x_1)\left(\cos\left(\pi x_1/2 + \pi\right) + 1\right) \\
g(x) &= 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \\
r(x_1) &= 5 + 10(x_1 - 0.5)^2 + \frac{1}{K}\cos(2K\pi x_1) \cdot 2^{\frac{S}{2}} \\
0 \leq & x_i \leq 1 \qquad i = 1, 2, \ldots, n \ ,
\end{aligned} \tag{6.10}$$

where K is the parameter specifying the number of knees on the Pareto optimal front, S determines the skewness of the front, and n is the number of variables used in the problem. The optimal solution to the problem is obtained for $x_2 = x_3 = \ldots = x_n = 0$ and the Pareto optimal front is thus a function of x_1 alone.

The first scenario is characterized by having a K value of 1, and an S value of 0. In order to be able to compare the results with those obtained in Branke, Deb, Dierolf, and Osswald (2004), as many of the NSGA-II parameters used in that paper will be identical for the experiments performed here. This means that for the first scenario, a population size of $N = 200$ will be used and the number of generations τ_{EA} will be set at 10.

The second scenario that will be used has a K value of 4, and an S value of 1. However, despite the fact that it is desirable to be able to compare the results with Branke, Deb, Dierolf, and Osswald (2004), a population size of $N = 200$ will be used instead of the $N = 100$ used in Branke, Deb, Dierolf, and Osswald (2004) and a maximum number of generations $\tau_{EA} = 10$ will be used. The reason why the population is chosen at $N = 200$ is due to the fact that it is also desired to compare the results of the different scenarios without exploiting any prior knowledge of the Pareto optimal front when performing the experiments.

The third scenario is based on the DEB2DK problem which is concave at the edges of the Pareto optimal front and given by

$$\begin{aligned}
\min f_1(x) &= g(x)r(x_1)\sin\left(\pi x_1/2\right) \\
\min f_2(x) &= g(x)r(x_1)\cos\left(\pi x_1/2\right) \\
g(x) &= 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \\
r(x_1) &= 5 + 10(x_1 - 0.5)^2 + \frac{1}{K}\cos(2K\pi x_1) \\
0 \leq & x_i \leq 1 \qquad i = 1, 2, \ldots, n \ ,
\end{aligned} \tag{6.11}$$

which uses a K value of 4. This problem also have optimal solutions for $x_2 = x_3 = \ldots = x_n = 0$, thus making x_1 the sole parameter to determine the place of a solution along the Pareto optimal front. The population size that will be used for this scenario is $N = 200$ and the maximum number of generations is set at $\tau_{EA} = 15$.

A summary of the scenarios and their corresponding parameters are given in table 6.1.

Parameter description	Values			Designation
Scenario	1	2	3	-
Problem	DO2DK	DO2DK	DEB2DK	-
Number of knees	1	4	4	K
Skewness of front	0	1	-	S
Population size	200	200	200	N
Maximum generations	10	10	15	τ_{EA}

Table 6.1: The values used for three different problems in an attempt to emphasize on curvature on the Pareto optimal front.

With the fitness functions in place for the different scenarios, it is time to choose the remaining parameters for running NSGA-II. These parameters will be kept constant for all of the experiments performed.

Because of the number of experiments that must be performed when investigating the four different crowding methods for three different scenarios, it would require too much space if it was to be investigated using both real valued and binary representations, since it would double the number of experiments from 12 to 24. It is thus chosen to only perform the experiments using real valued representation, whereas experiments using binary representation will not be performed. This is also partly due to the fact that experiments using binary representations takes much longer to perform and for the large amount of variables used the binary representation is expected to take much longer to converge on the Pareto optimal front than a corresponding real valued representation.

The parameters used for the experiments are summarized in table 6.2 on the next page.

Once again, the parameters used in the algorithm are those used by NSGA-II per default.

6.3 Experimental Results

In order to fully investigate whether the proposed methods are better at emphasizing the curved regions of the Pareto optimal front than the regular NSGA-II crowding method, the experiments are also performed for the original NSGA-II crowding. The results from the previous chapter concerning the scaling issues for the fitness functions will be incorporated into the experiments and as such, all of the experiments, including the original NSGA-II crowding, will be subjected to the method of local scaling of the individual fronts before any of the proposed crowding measures are applied. So, with the preliminary issues in place let us begin with a look at how the different methods perform for scenario 1.

Parameter description	Type	Value	Designation
Number of fitness functions	Functions	2	f_1, f_2
Number of variables	Reals	30	$x_1, \ldots, x_{30}$
$x_1, \ldots, x_{30}$	Real	$[0, 1]$	$[x_{min}, x_{max}]$
Selection operator	Tournament	2	s_{size}
Crossover operator	SBX	10	-
Crossover probability	-	0.9	p_c
Mutation operator	Polynomial	50	-
Mutation probability	-	0.03	p_m
Endpoint parameter	Real	10^{-5}	ϵ
Random seed	-	0.1234	r_{seed}

Table 6.2: The parameters used in NSGA-II for the experiments emphasizing curvature on the Pareto optimal front and using real valued variables.

6.3.1 Scenario 1

As described previously, scenario 1 is based on the DO2DK problem containing a single knee and no skewness of the Pareto optimal front. It was desired to run the algorithm for 10 generations since it would make the result comparable to those given in Branke, Deb, Dierolf, and Osswald (2004). However, as can be seen from figure 6.8 the non-dominated set obtained at generation 10 has not at all converged to the set of Pareto optimal solutions.

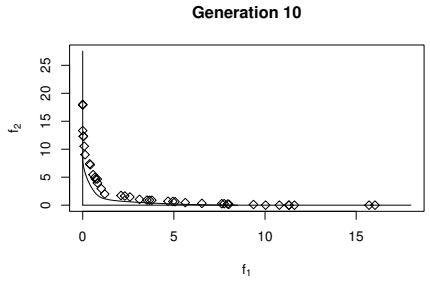
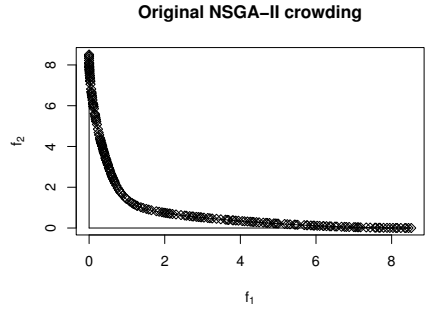


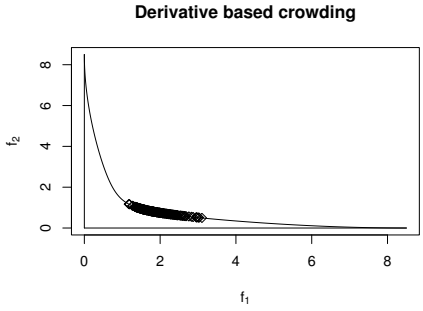
Figure 6.8: The set of non-dominated solutions after 10 generations when running NSGA-II on scenario 1 has not converged to the set of Pareto optimal solutions.

At this point, it was decided to run all of the experiments for at least 50 generations, such that it would be possible for the population to reach the Pareto optimal front and allowing for the proposed crowding methods to take effect.

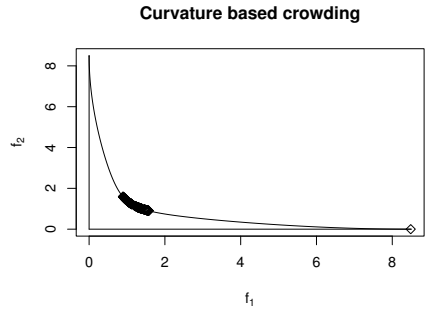
By running the algorithm for 50 generations, the results illustrated in figure 6.9 on the facing page was obtained.



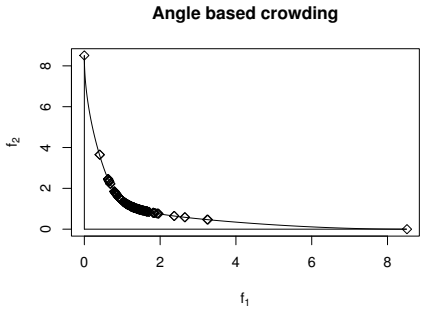
(a) Original NSGA-II crowding.



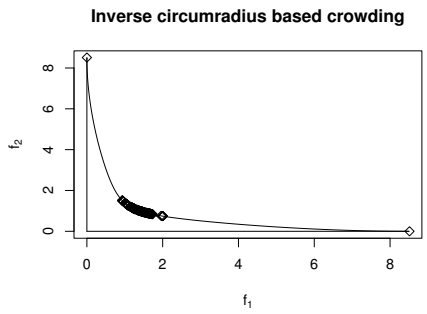
(b) Derivative based crowding.



(c) Curvature based crowding.



(d) Angle based crowding.



(e) Inverse circumradius based crowding.

Figure 6.9: Pareto optimal fronts after 50 generations for different crowding measures applied to scenario 1.

It is clearly shown that the original NSGA-II crowding shown in figure 6.9a is capable of producing a nice uniform distribution of points, but does not emphasize in the regions with the most tradeoff. When it comes to the results of the crowding method based on the estimated first derivative shown in figure 6.9b it is clear that the method is capable of emphasizing the region of the Pareto front where the slope is close to 1 and also a bit further in the direction of fitness function f_1. The bias towards f_1 produced by this estimated first derivative cannot be explained since no such bias was used in the crowding measure. However, it is also seen that the method is not capable of maintaining any points close to the endpoints and as such the method does not give an accurate picture as to the extent of the Pareto optimal front.

In figure 6.9c, the curvature based crowding method is successful at emphasizing the region with highest curvature. The method does not yield any solutions near one of the endpoints and as such this method also fails in giving an accurate picture of the actual front since it cannot preserve solutions near both of the endpoints of the Pareto optimal front.

When it comes to the angle based measure in figure 6.9d, it is very good at both emphasizing the region of high curvature while also preserving the endpoints of the Pareto optimal front. It even preserves some solutions part of the way from the region with most curvature up towards the endpoints such that an interpolation of the Pareto optimal front can easily be found despite the sparsely distributed points in that area.

For the crowding measure using the inverse circumradius in figure 6.9e, the curvature is again emphasized, and the distribution of points is actually much similar to that of the curvature based crowding in figure 6.9c, except that the inverse circumradius method is capable of preserving solutions at both endpoints of the Pareto optimal front whereas the curvature based method could only preserve one of the endpoints.

For scenario 1 it has been shown that all of the proposed methods were capable of emphasizing the region of most curvature over the rest of the Pareto optimal front in one way or the other, however when looking at the results objectively, the method that gave the most promising results were that using the angle based crowding measure since it was also capable of preserving solutions at the endpoints and also somewhat in the intermediate regions.

When comparing the obtained results with those presented in Branke, Deb, Dierolf, and Osswald (2004) it is interesting to note how different the results of that paper is from those obtained here. The reason for this difference cannot be explained fully, but it is expected that the increased number of generations used for obtaining the results in this thesis might be the cause of some of the differences.

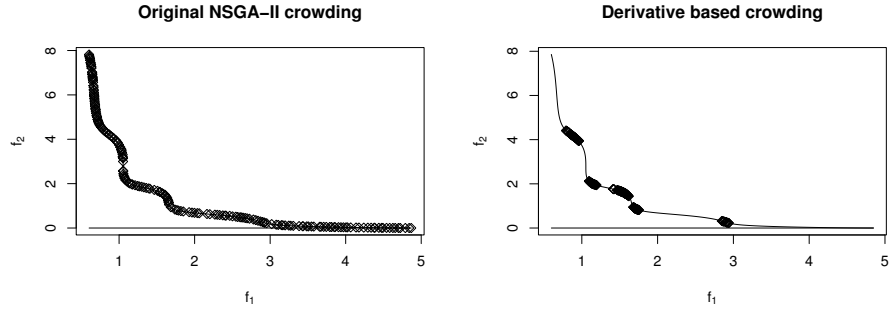
Having presented the results of scenario 1, it is now time to take a look at the results obtained using the different crowding schemes for scenario 2.

6.3.2 Scenario 2

The results obtained when running NSGA-II on scenario 2 using the proposed crowding methods are shown in figure 6.10 on the facing page.

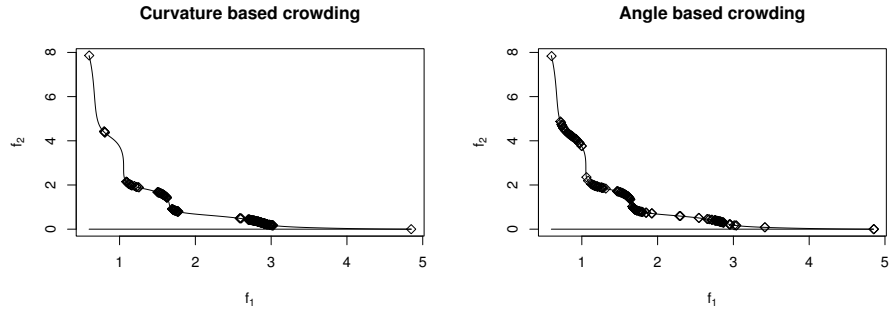
As expected for the original NSGA-II crowding in figure 6.10a, a near uniform distribution of points has been achieved without emphasis on any of the regions of the Pareto optimal front.

By taking a look at figure 6.10b where the crowding method used was that of the estimated first



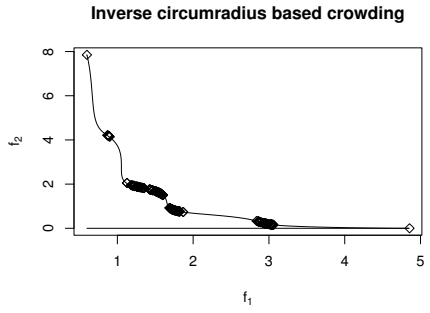
(a) Original NSGA-II crowding.

(b) Derivative based crowding.



(c) Curvature based crowding.

(d) Angle based crowding.



(e) Inverse circumradius based crowding.

Figure 6.10: Pareto optimal fronts after 50 generations for different crowding measures applied to scenario 2.

derivative, it can be seen that the method is successful at emphasizing the regions of the Pareto optimal front having the largest curvature. As was the case for scenario 1, the derivative based crowding method once again is not capable of maintaining any solutions in the endpoints and does not give a fully accurate representation of the Pareto optimal front.

In figure 6.10c, the crowding method using the curvature based crowding measure is once again capable of emphasizing some of the regions of highest curvature. It seems like one of the regions is underrepresented which might be caused by the fact that the scaling of the figure is not even on both axes. In fact, the region that is underrepresented does not have as much curvature as the figure would indicate and as such there is no bias towards any of the other regions with high curvature. In contrast to the obtained result for scenario 1, the curvature based crowding method for scenario 2 is able to preserve the solutions at the endpoints of the Pareto optimal front.

The angle based crowding applied to scenario 2 as seen in figure 6.10d is capable of nicely emphasizing nicely the regions with most curvature. Once again, it does not only preserve the endpoints of the Pareto optimal front, but it also has a few intermediate solutions such that the shape of the Pareto optimal front is quite evident even if it was not for the front drawn on the figure separately.

The crowding method based on the inverse circumradius the result shown in figure 6.10e indicates that this method again is capable of emphasizing on the regions having the most curvature. As was the case for the curvature based crowding for scenario 2, the method using the inverse of the circumradius does not maintain too many solutions in one of the regions, but as explained before this is mostly due to the uneven scaling used in the figures. However, it does look like this method does not maintain the exact regions of curvature but actually have biased the regions a little toward fitness functions f_1. The reason for this is not known and cannot be explained as being caused by the use of the crowding method in question.

When comparing the results of scenario 2 with the corresponding results of Branke, Deb, Dierolf, and Osswald (2004), the two methods that produce results most similar to the angle based crowding measure of that paper are actually the curvature based crowding scheme and the inverse circumradius based crowding scheme. For this scenario it is, however, difficult to make direct comparisons of the results obtained here and those mentioned in the paper since the methods here also find solutions in the concave regions of the Pareto optimal front, whereas the method applied in the paper only found solutions in the convex regions of the Pareto optimal front.

Since we now have presented the results from scenario 2, the only remaining scenario to investigate is scenario 3.

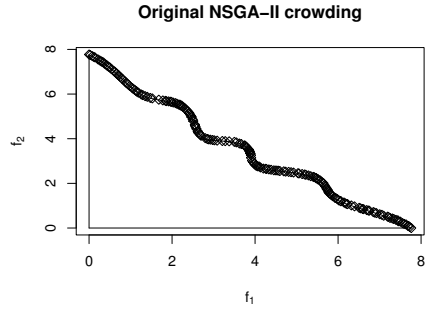
6.3.3 Scenario 3

The results of running the proposed crowding methods on scenario 3 are presented in figure 6.11 on the facing page.

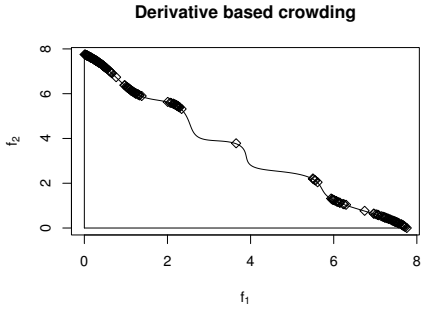
As expected, the original NSGA-II crowding measure maintains a uniform distribution of points along the entire Pareto optimal front as can be seen in figure 6.11a.

In figure 6.11b, the derivative based crowding method is seen to emphasize on the regions of curvature near the edges of the Pareto optimal front whereas the curved regions at the center of the front are very sparsely represented.

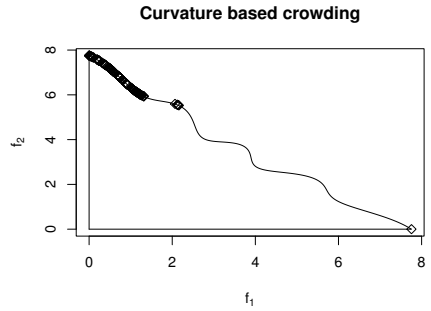
For both the curvature based crowding method in figure 6.11c and the inverse circumradius in



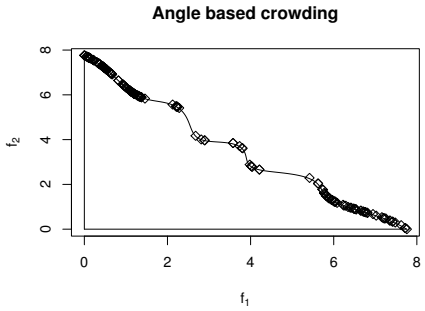
(a) Original NSGA-II crowding.



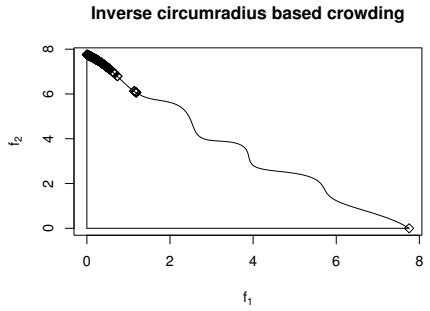
(b) Derivative based crowding.



(c) Curvature based crowding.



(d) Angle based crowding.



(e) Inverse circumradius based crowding.

Figure 6.11: Pareto optimal fronts after 50 generations for different crowding measures applied to scenario 3.

figure 6.11e only the curved region near one of the edges are emphasized whereas the other regions along the front are not represented. Both methods do, however, find both endpoints and preserve them.

Finally, the angle based crowding measure in figure 6.11d is capable of emphasizing the curved regions along the entire front. Notably, the regions near the edges are emphasized more than the center region of the Pareto optimal front.

From the results it is clear that the method which performed best for scenario 3 is the angle based measure since it was the method that was best able to emphasize the curved region along the entire Pareto optimal front. It is also evident from the results for this scenario that the proposed methods did not produce solution sets with a good distribution of points when it comes to emphasizing curvature.

One of the main factors that might have had a crucial role in the performance of the crowding measures for this and possibly also the previous scenarios has been the issue of drifting. Since all of the proposed methods only emphasized curvature and did not have any preference when it came to areas having the same amount of curvature, this might have influenced the results considerably. Especially for scenario 3 it is evident that it might have had an influence. This is why it was decided to perform a series of experiments using scenario 3 where each of the proposed crowding methods were combined with the regular NSGA-II crowding measure. By doing so, it is hoped that the different crowding methods would be capable of maintaining solutions over larger areas of the Pareto optimal front. Thus, the following section will present the results of using a hybrid combination of the proposed crowding methods and the original NSGA-II crowding measure for scenario 3.

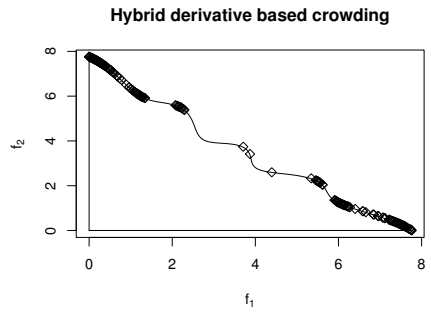
6.3.4 Hybrid Crowding Measures

Because of the issue raised in the previous section, it was decided to perform some additional experiments on scenario 3 using a hybrid version of the crowding measures. The way those hybrid crowding measures are implemented is by calculating both the original crowding measure $d_{j_{orig}}$ and the proposed measure $d_{j_{prop}}$ during each experiment. The crowding measures are then normalized to lie in the interval 0 to 1 and then added together. This can also be written as

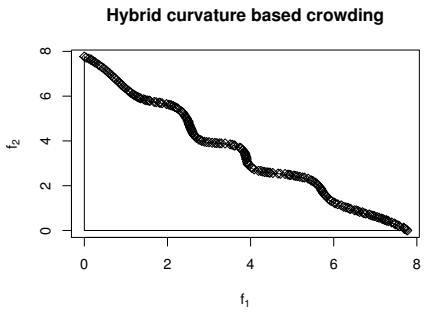
$$d_j = \frac{d_{j_{orig}} - \min_{i \in N}(d_{i_{orig}})}{\max_{k \in N}(d_{k_{orig}}) - \min_{i \in N}(d_{i_{orig}})} + \frac{d_{j_{prop}} - \min_{l \in N}(d_{l_{prop}})}{\max_{m \in N}(d_{m_{prop}}) - \min_{l \in N}(d_{l_{prop}})} \quad . \tag{6.12}$$

With this hybrid crowding measure, it was then possible to obtain the results shown in figure 6.12 on the facing page.

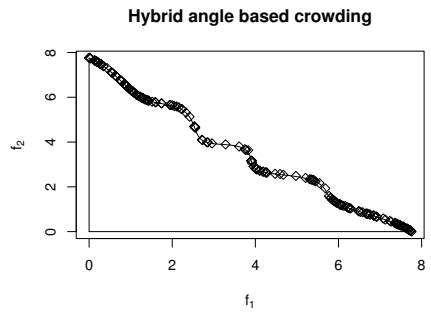
The results show that there is not much of an improvement for the hybrid derivative based crowding in figure 6.12a. However, the fact that it is slightly improved did result in further experiments to be performed. These showed that if the original crowding measure was given a higher weighting, then the distribution also improved. The incorporation of the original crowding measure thus meant that the hybrid derivative based crowding measure became capable of finding and preserving the edges of the Pareto optimal front. Since tuning of such a weighting would require prior knowledge about the problem, it turns out that the method would not help much in automatically



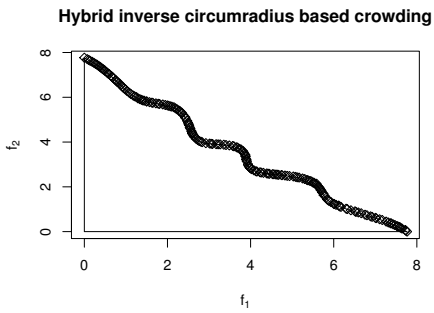
(a) Hybrid derivative based crowding.



(b) Hybrid curvature based crowding.



(c) Hybrid angle based crowding.



(d) Hybrid inverse circumradius based crowding.

Figure 6.12: Pareto optimal fronts after 50 generations for different equally weighted hybrid crowding measures applied to scenario 3.

achieving a good distribution of points. The method would only shift the problem into another setting.

For the hybrid curvature based and hybrid inverse circumradius based methods as shown in figures 6.12b and 6.12d, it is evident that the original crowding measure is overly emphasized and the results are more or less uniformly distributed solutions. The reason for this is due to the exponential structure of the curvature and inverse circumradius crowding methods, which can result in large crowding distances, but only in a few places. Since these large crowding distances are only obtainable in a few regions on the Pareto optimal front, the majority of solutions will have much smaller crowding distances. Since the hybrid method uses normalization before adding the two different crowding measures, this means that for many of the individuals, the original crowding measure will be dominant. In order for these two hybrid methods to be successful, it is thus necessary to tune the weighting of the crowding measures specifically to the problem since it depends highly on the shape of the Pareto optimal front.

For the hybrid angle based crowding method in figure 6.12c, the solutions have become somewhat more spread out along the front, while still emphasizing the regions having the most curvature. It might be the same issue that affected the hybrid curvature based and inverse circumradius based methods which also affects the angle based method. However, the reason why it might not be affected as much could be because the normalization actually could not severely reduce the importance of any of the angle based crowding measures as the values per default could not exceed 1 since that was the limitation set by using the cosine of the angle in the crowding measure.

With this extensive discussion of the results obtained when running NSGA-II on different problems using different crowding measures, it is now time to draw some conclusions based on these results.

6.4 Concluding Remarks

In this chapter, several different ways of performing crowding that should emphasize on the regions of the Pareto optimal front with most curvature have been presented. In total, four distinct methods were proposed and investigated. These methods were then also used in hybrid versions where the original crowding measure of NSGA-II was included as well.

Common to all of the proposed methods was the fact that they did not include any sort of penalties for multiple solutions located at the same location. Thus, even though the different methods were more or less successful in their own distinct way, they all suffered from the effects of genetic drift. To accommodate for this, the hybrid versions were introduced.

In general, the derivative based crowding method performed adequately when it was used as the sole crowding measure. It was not the best crowding measure, mostly due to the drawback that it could not find and preserve the endpoints of the Pareto optimal front. However, once the hybrid derivative based crowding method was introduced, this issue was resolved. Though not presented visually, the hybrid derivative based crowding measure became better once the weighting of the original crowding measure was increased. It is thus possible to use this method for finding the regions of the Pareto optimal front with the most curvature, but in order to use it to its fullest potential it would be necessary to find the optimal weighting between the derivative and the original crowding measure. Since such a weighting could easily turn out to be problem dependent, the

method is thus not without drawbacks.

Two of the methods turned out to be quite similar in their performance, namely the curvature based crowding measure and the inverse circumradius crowding measure. On their own, they performed relatively well for scenario 1 and 2 where they could emphasize the curved regions and also find and preserve the endpoints of the Pareto optimal front. Though, when it came to scenario 3, both methods turned out to have problems at finding all of the curved regions. For the hybrid versions of these two crowding measures it turned out that they became totally dominated by the original crowding measure, and resulted in more or less uniformly distributed solutions along the Pareto optimal front. The likely reason for this problem was identified as being the high degree of non-linearity, which resulted in the contribution of the two crowding measures becoming negligible and thus overshadowed by the original NSGA-II crowding measure. As such, the use of these two methods cannot be recommended since they turned out to produce inconsistent results.

The angle based crowing method turned out to consistently produce good results where the curved regions of the Pareto front were emphasized while also preserving some intermediate solutions giving better insight to the problem at hand. Both the regular version and the hybrid produced usable results, but if used, the hybrid version is recommended since it would not suffer from any drifting issues which might influence the pure angle based crowding measure.

The inherent limitation of all of the proposed methods has been that they could only be applied using two fitness functions at a time. Thus, the use of the proposed methods should be either applied to a case having only two fitness functions or be applied to pairwise combinations of fitness functions. When applied pairwise, the resulting computational load would be of the order $O(M^2 N \log N)$ since all of the different combinations of fitness functions should be taken into consideration. For comparison, the original crowding measure of NSGA-II is of the order $O(MN \log N)$. However, if applying the crowding measures to more than two fitness functions, some additional considerations should be taken into account, such as the fact that the angles could attain all values in the range 0° to 360° which might interfere with the methods of angle based crowding and inverse circumradius based crowding due to the use of trigonometric functions. This also means that for the cases using more than two fitness functions, the proposed methods might not be too usable. This is what has lead to the issue that will be discussed in the next chapter, namely a crowding scheme based on projections that can be applied to a problem of any dimension.

Chapter 7

Projection Based Crowding

In light of the limitations of the proposed crowding methods in the previous chapter, another and more general approach to performing the crowding calculations, such that they can emphasize on different regions of the Pareto optimal front, will be formulated in this chapter. The crowding method presented in this chapter will be based on projections, and as such it should be applicable to problems of any dimension. In Branke and Deb (2004), such a projection based crowding method was used to bias the distribution of points on the Pareto optimal front based on input from the user. However, since they focused the research in that paper at integrating user preference into the optimization process, the crowding measure was only applied to problems that did not contain knees or ankles, and it is not possible to conclude whether the method would be useful for such a purpose or not. First, let us begin with a description of the principle used for the projection based crowding measure.

7.1 Principle of Projection Based Crowding

As indicated by the name, the projection based crowding scheme uses projections of the solutions on the Pareto optimal front in order to make the crowding calculations. As such, it is necessary to perform a projection of the solutions on the Pareto optimal front, similar to what is illustrated in figure 7.1 on the next page.

The vector $\vec{u_j}$ is comprised of the fitness values for an individual and it is desired to find the projection $\vec{v_j}$ of this on a hyperplane, which is defined using a normal vector $\vec{\eta}$. For a two-dimensional problem, such a hyperplane would be a line, and for a three-dimensional problem the hyperplane would be a regular plane. The vector $\vec{v_j}$ is given by

$$\vec{v_j} = \vec{u_j} - \vec{u_{j_\eta}} \quad , \tag{7.1}$$

which using the fact that

$$\vec{u_{j_\eta}} = \vec{\eta} \cdot \frac{\langle \vec{u_j}, \vec{\eta} \rangle}{\|\vec{\eta}\|^2} \quad , \tag{7.2}$$

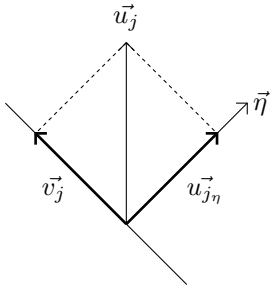


Figure 7.1: Illustration of the way a solution $\vec{u_j}$ can be projected onto a hyperplane specified by a normal vector $\vec{\eta}$ and produce the projection $\vec{v_j}$.

results in

$$\vec{v_j} = \vec{u_j} - \vec{\eta} \cdot \frac{\langle \vec{u_j}, \vec{\eta} \rangle}{\|\vec{\eta}\|^2} \quad . \tag{7.3}$$

Using the normal vector for the projection hyperplane, it is possible to calculate the projected fitness values of an individual. This projected version of the fitness values can then be used as a basis for calculating another set of crowding distances $\hat{d_j}$ using the original NSGA-II crowding measure of equation (2.13). This crowding measure is then used in combination with the original crowding measure $d_{j_{orig}}$ in the same way as used in Branke and Deb (2004), resulting in the expression given by

$$d_j = d_{j_{orig}} \left(\frac{\hat{d_j}}{d_{j_{orig}}} \right)^{\beta} \quad . \tag{7.4}$$

The β values allow for the difference between the two crowding distances to be emphasized more or less. However, for specific values of β such as 0 or 1, the crowding measure will actually correspond to the pure crowding measure of the original NSGA-II crowding $d_{j_{orig}}$ or to the crowding measure based solely on the projected values $\hat{d_j}$ respectively. An illustration of the projected individuals and the crowding distances obtained is shown in figure 7.2 on the facing page.

When the projection based crowding method will be investigated in the following, several different values for β and a number of different $\vec{\eta}$ vectors will be tested on the 3 scenarios given in the previous chapter.

Before continuing to the experiments, there is one issue that is expected to give problems with regard to the proposed crowding method. The issue arises because of the way the crowding calculations of NSGA-II is performed, namely using hypercubes. The hypercubes are calculated as half of the perimeter of the cuboid which is spanned by closest neighbors to each side of an individual $j-1$ and $j+1$ as illustrated in figure 2.37 on page 50. This actually corresponds to calculation of the distance between those individuals $j-1$ and $j+1$ using the Manhattan distance metric. Now, because of this metric, there are cases where it is expected that the method could fail in emphasizing any particular regions of the Pareto optimal front. The problem is easiest explained using illustrations. In figure 7.3 on the facing page, it is seen how two different orientations of $d_{j_{orig}}$ will result in identical $\hat{d_j}$ values.

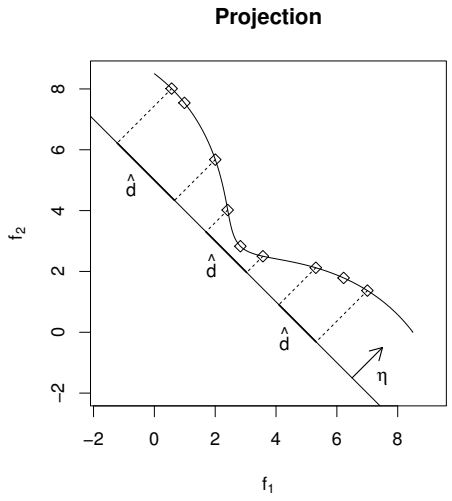
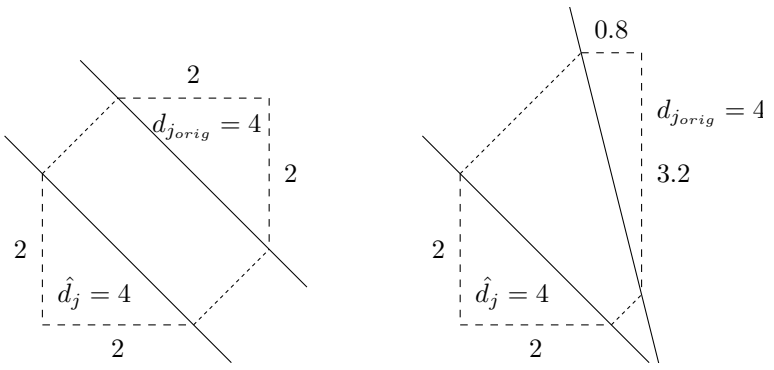


Figure 7.2: An illustration of projection of the fitness values onto a hyperplane given by the normal vector $\vec{\eta}$.



(a) Parallel. (b) Non-parallel.

Figure 7.3: An example of two different situations where the crowding distances based on projection and original crowding measures are either (a) parallel to each other or (b) not parallel to each other when using Manhattan distance metric.

Since the two differently oriented $d_{j_{orig}}$ distances gave rise to identical $\hat{d}_j$ values, the proportional relationship $\frac{\hat{d}_j}{d_{j_{orig}}}$ will also be identical. Thus, the desired emphasis on regions with large trade-offs or curvature cannot be achieved. This issue will only occur for the cases when $\vec{\eta}$ emphasizes each fitness function equally, but since it is expected that such a vector $\vec{\eta}$ will often be used, the issue must be addressed.

The issue is expected to be resolved by using a Euclidean metric for the distance calculations, such that the original NSGA-II crowding applied to each of the distance measures are replaced with the Euclidean equivalent given by

$$d_j = d_j + \left(\frac{f_{i,j+1} - f_{i,j-1}}{f_{i,max} - f_{i,min}} \right)^2 \quad . \tag{7.5}$$

From figure 7.3 on the page before it is also clear that when using the Euclidean based distance metric the relationship $\frac{\hat{d}_j}{d_{j_{orig}}}$ will correspond to cosine to the angle between $\hat{d}_j$ and $d_{j_{orig}}$.

So, when the experiments are conducted, they will be tested using both the original Manhattan based crowding distance and the Euclidean based crowding distance such that potential issues due to the use of the Manhattan metric can be resolved.

With all these preparatory issues in place, it is now time to take a look at the experimental setup.

7.2 Experimental Setup

The fitness functions used for testing of this projection based crowding measure for emphasizing curvature on the Pareto optimal front will be the same as those used in the previous chapter. Similarly, the scenarios used in chapter 6 on page 105 will be the same and the details are summarized in table 7.1.

Parameter description	Values			Designation
Scenario	1	2	3	-
Problem	DO2DK	DO2DK	DEB2DK	-
Number of knees	1	4	4	K
Skewness of front	0	1	-	S
Population size	200	200	200	N
Maximum generations	50	50	50	τ_{EA}

Table 7.1: The values used for three different problems in an attempt to emphasize on curvature on the Pareto optimal front using projection based crowding.

For each of the scenarios a series of experiments for different combinations of β, $\vec{\eta}$, and whether to use Manhattan or Euclidean distance metric, will be conducted. The β values that will be tested are 0 (original NSGA-II crowding), 1 (pure projection based crowding), and 100 (emphasizing differences between $d_{j_{orig}}$ and $\hat{d}_j$). For the normal vector $\vec{\eta}$ the different values that will be used

are $\vec{\eta} = (1, 1)$, $\vec{\eta} = (1, 0)$, and $\vec{\eta} = (0, 1)$. Since all of these different combinations of values generates a number of 18 experiments to be performed for each scenario, some selection will be done so as to avoid presenting too many irrelevant results. An example of such would be for $\beta = 0$ and any combination of $\vec{\eta}$ values, since the normal vector will not be relevant when the projection term is disregarded.

The remainder of the parameters used for the experiments are identical to the ones used for the experiments in chapter 6 on page 105. The parameters are summarized in table 7.2.

Parameter description	Type	Value	Designation
Number of fitness functions	Functions	2	f_1, f_2
Number of variables	Reals	30	$x_1, \ldots, x_{30}$
$x_1, \ldots, x_{30}$	Real	$[0, 1]$	$[x_{min}, x_{max}]$
Selection operator	Tournament	2	s_{size}
Crossover operator	SBX	10	-
Crossover probability	-	0.9	p_c
Mutation operator	Polynomial	50	-
Mutation probability	-	0.03	p_m
Random seed	-	0.1234	r_{seed}

Table 7.2: The parameters used in NSGA-II for the experiments emphasizing curvature on the Pareto optimal front using real valued variables and projection based crowding.

With the details of the algorithm in place, it is time to take a look at the results obtained.

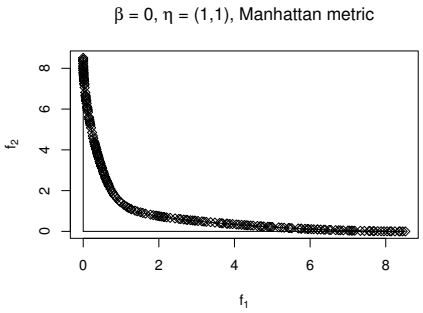
7.3 Experimental Results

When performing the experiments, the method of local scaling proposed in chapter 5 on page 83 will once again be applied to all fitness values before the distance measures are calculated. This ensures that all objectives will have an identical level of influence before the emphasis will be put on the different regions of the Pareto front. Now, let us first take a look at the results obtained for scenario 1.

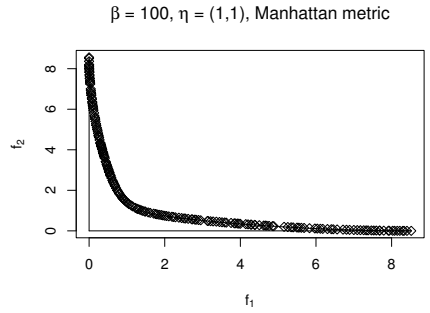
7.3.1 Scenario 1

The first set of results presented for scenario 1 will focus on the difference between using the Manhattan based crowding measure and the Euclidean equivalent. For β values of 0 and 1 and an $\vec{\eta}$ value of $\vec{\eta} = (1, 1)$, the results were indistinguishable when using either metric. However, when it came to using a β value of 100, there are interesting differences. In figure 7.4 on the next page, the results obtained when using the two different distance metrics are shown along with the result obtained for the original NSGA-II crowding measure.

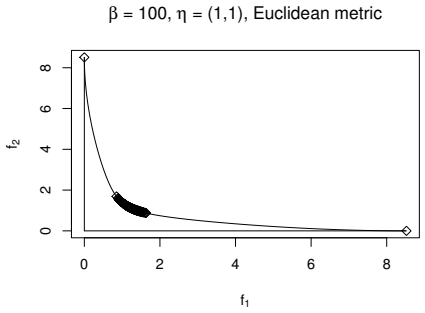
It is clear that for the Manhattan based metric the difference between using a β value of 100 (figure



(a) Original crowding using a β value of 0.



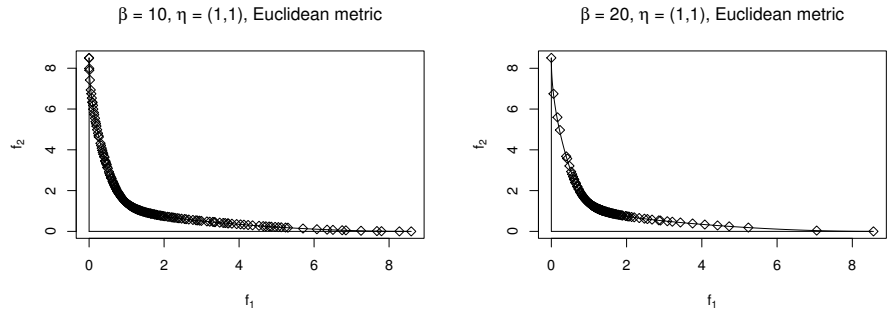
(b) Manhattan based crowding for large β value.



(c) Euclidean based crowding for large β value.

Figure 7.4: Pareto optimal fronts showing the difference between Manhattan and Euclidean based crowding measures after 50 generations for projection based crowding applied to scenario 1.

7.4b) and the original NSGA-II crowding (figure 7.4a) are minute. However, when looking at the results obtained for the Euclidean based metric and a β value of 100 (figure 7.4c), the result is very different. Suddenly, the solutions have converged to the region of the Pareto optimal front where the region has the biggest curvature. Noticeably, the result using the Euclidean based metric also preserves the endpoints as well, despite having such a large concentration of solutions in one specific region. The obtained results thus confirm the suspicions mentioned previously that the Manhattan based metric was unsuitable when using the projection based crowding measure. Because the result for $\beta = 100$ showed that the solutions became highly concentrated in a small region, it was decided to perform some additional experiments using the Euclidean based metric and β values of 10 and 20. The result of that experiment is shown in figure 7.5.



(a) Euclidean metric with a β value of 10.

(b) Euclidean metric with a β value of 20.

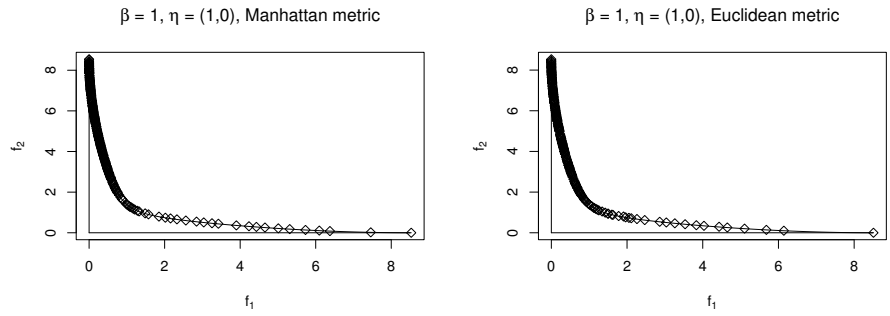
Figure 7.5: Pareto optimal front after 50 generations for projection based crowding applied to scenario 1 and using β values of 10 and 20.

It is clear that those intermediate values of β are good at partially emphasizing the region of the Pareto optimal front with the highest degree of curvature while still preserving a small number of solutions along the rest of the front.

The effect of using a different set of values for $\vec{\eta}$, namely $\vec{\eta} = (1, 0)$ is shown in figure 7.6 on the next page.

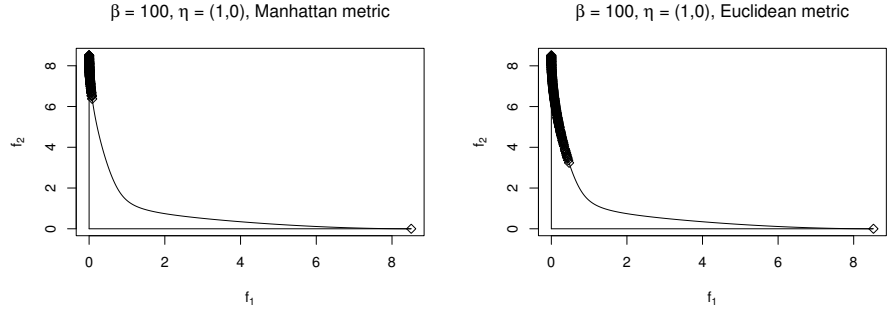
In the figure, it is evident that both the Manhattan based metric and the Euclidean based metric are capable of shifting the solutions along the Pareto optimal front to emphasize solutions along f_2. The Manhattan based metric tends to crowd the solutions more tightly together than the Euclidean based metric, which is seen most clearly for $\beta = 100$ (figure 7.6c and 7.6d). It is also worth to note that for this set of experiments, a β value of 1 (figure 7.6a and 7.6b) already shows a shift towards f_2, whereas the case for $\vec{\eta} = (1, 1)$ did not produce a result much different from the original crowding measure. For all of the experiments, the endpoints once again were successfully preserved.

Now, for $\vec{\eta} = (0, 1)$ a similar result as the one above was achieved, with the only difference being that for this case it was along f_1 that the solutions were emphasized. The results for this case



(a) Manhattan metric for a β value of 1.

(b) Euclidean metric for a β value of 1.

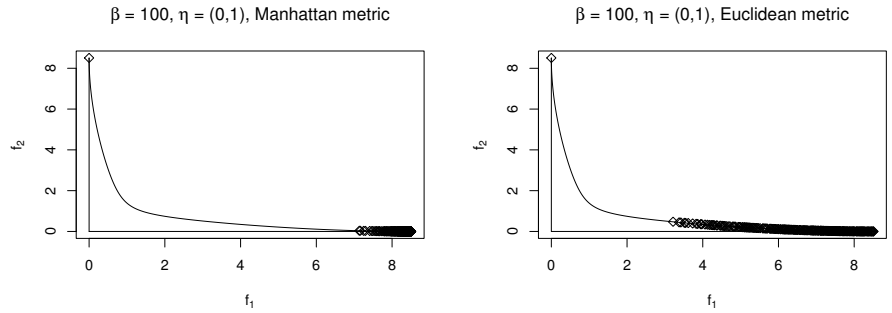


(c) Manhattan metric for a β value of 100.

(d) Euclidean metric for a β value of 100.

Figure 7.6: Pareto optimal fronts after 50 generations for projection crowding applied to scenario 1 using $\vec{\eta} = (1, 0)$.

using a β value of 100 are shown in figure 7.7.



(a) Manhattan metric for a β value of 100 and $\vec{\eta} = (0, 1)$.

(b) Euclidean metric for a β value of 100 and $\vec{\eta} = (0, 1)$.

Figure 7.7: Pareto optimal fronts after 50 generations for projection based crowding applied to scenario 1 using $\vec{\eta} = (0, 1)$ and a β value of 100.

With the results of scenario 1 presented, we now move on to take a look at how the projection based crowding scheme performed for scenario 2.

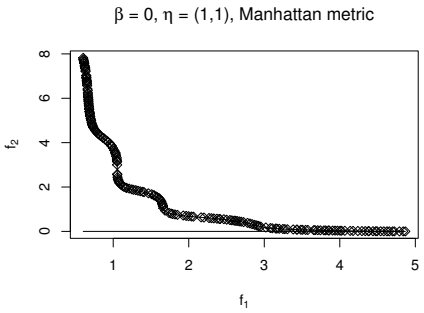
7.3.2 Scenario 2

Similar to the situation encountered for scenario 1, when comparing the results for using the two different distance metrics, it turns out that the Manhattan based metric is incapable of emphasizing any regions of the Pareto optimal front when using $\vec{\eta}$ values of $\vec{\eta} = (1, 1)$. This is shown in figure 7.8 on the next page where, once again, the result obtained using the original NSGA-II crowding method is included for comparison.

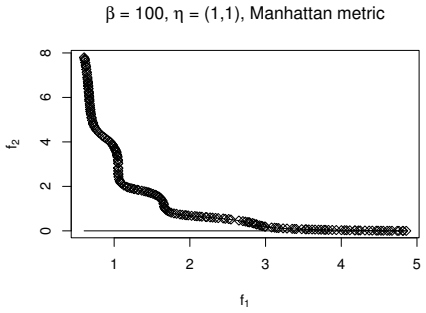
Once again, a set of intermediate values of $\beta = 10$ and $\beta = 20$ was used to further investigate the distribution of solutions along the Pareto optimal front when the difference between the two crowding measures, original and projected, are not overly emphasized. The results obtained for running those experiments are shown in figure 7.9 on page 137.

The figure shows that the intermediate values of β will give a coverage of the Pareto optimal front which emphasizes on the regions with most curvature, but also preserves a scarce number of solutions in the remaining regions. This makes it easy to interpolate the shape of the Pareto optimal front, while still having the desired emphasis on curvature.

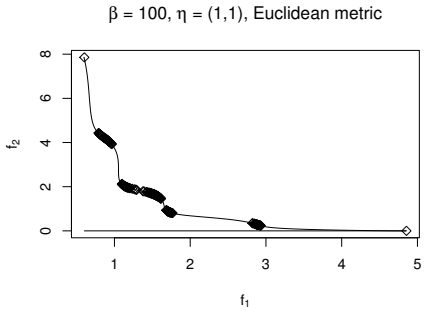
When it comes to using different values for $\vec{\eta}$, the only results that will be presented are those obtained using $\vec{\eta} = (1, 0)$, since the results obtained for $\vec{\eta} = (0, 1)$ are so similar with the only exception being that the emphasis is put on the other fitness function. So, for $\vec{\eta} = (1, 0)$, the results obtained are shown in figure 7.10 on page 138.



(a) Original crowding using a β value of 0.

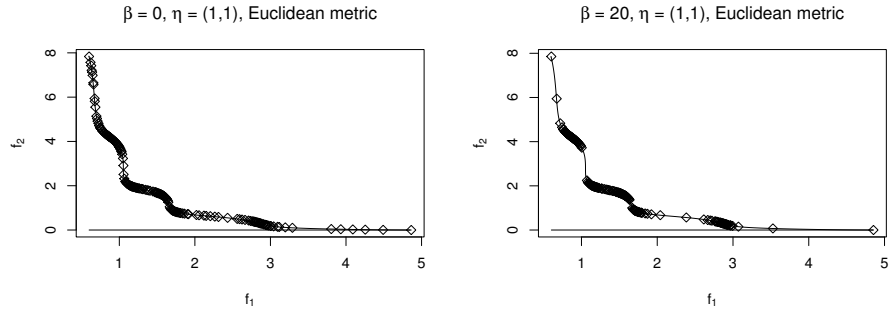


(b) Manhattan based crowding for large β value.



(c) Euclidean based crowding for large β value.

Figure 7.8: Pareto optimal fronts showing the difference between Manhattan and Euclidean based crowding measures after 50 generations for projection based crowding applied to scenario 2.



(a) Euclidean metric with a β value of 10.

(b) Euclidean metric with a β value of 20.

Figure 7.9: Pareto optimal front after 50 generations for projection based crowding applied to scenario 2 and using β values of 10 and 20.

The results are once again as expected, since the solutions were shifted towards only one of the fitness functions while still preserving the endpoint solutions. Because the problem for scenario 2 contained several knees and thus also had several regions which would be more or less parallel to each of the fitness functions, it can be seen that the solutions are distributed such that they cover most of those areas, whereas the rest of the Pareto optimal front does not contain any solutions except for the endpoints as mentioned above.

Continuing on, it is now time to take a look at the results obtained for scenario 3.

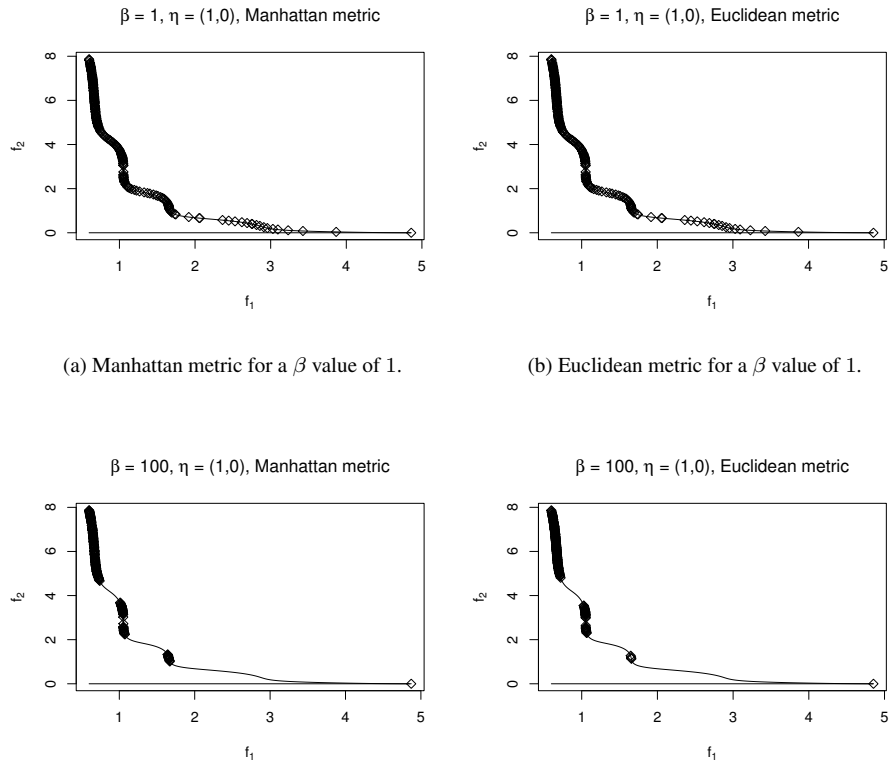
7.3.3 Scenario 3

For scenario 3, it is no surprise that the Manhattan based metric once again was unable to emphasize any particular regions of the Pareto optimal front when using $\vec{\eta} = (1, 1)$. This is shown in figure 7.11 on page 139.

In figure 7.11b, it is, however, quite remarkable that the distribution is focused mainly near the edges and that the sections with tradeoffs in the middle section of the front tend to have a scarce distribution of solutions. The reason for this might be caused by the fact that in the areas near the endpoints there are larger regions which have slopes that are parallel to the slope of the hyperplane given by $\vec{\eta}$. In the middle region, the areas having that same slope are much smaller and it is thus difficult for the algorithm to maintain those solutions.

This issue can, however, be solved by not emphasizing the difference between the two crowding measures as much. Once again, some experiments were performed using β values of 10 and 20 and the results are shown in figure 7.12 on page 139.

In those figures, it is evident that by reducing the pressure for emphasizing the difference between the crowding measures, the distribution of points becomes quite nicely distributed.



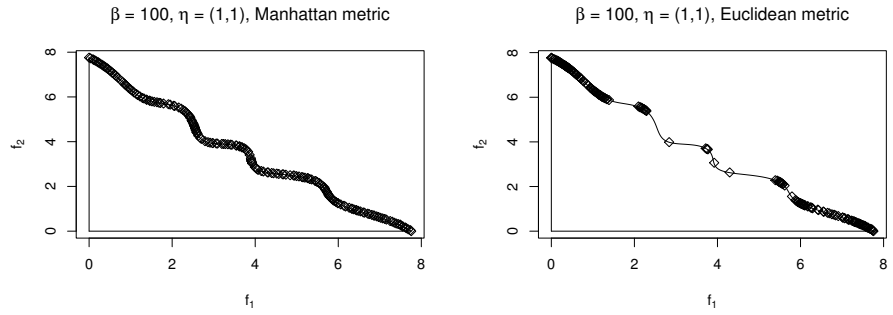
(a) Manhattan metric for a β value of 1.

(b) Euclidean metric for a β value of 1.

(c) Manhattan metric for a β value of 100.

(d) Euclidean metric for a β value of 100.

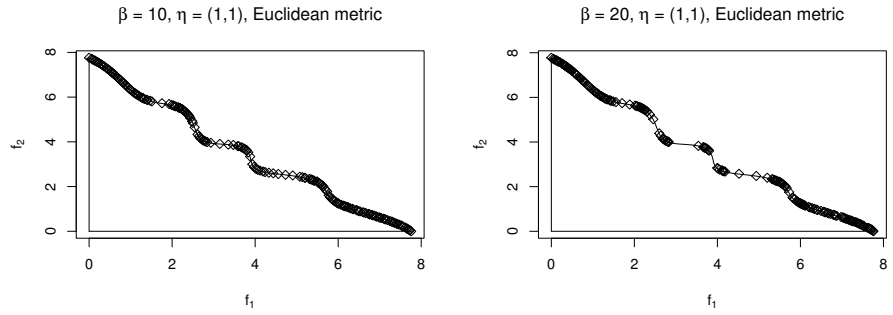
Figure 7.10: Pareto optimal fronts after 50 generations for projection based crowding applied to scenario 2 using $\vec{\eta} = (1, 0)$.



(a) Manhattan based crowding for large β value.

(b) Euclidean based crowding for large β value.

Figure 7.11: Pareto optimal fronts showing the difference between Manhattan and Euclidean based crowding measures after 50 generations for projection based crowding applied to scenario 3.

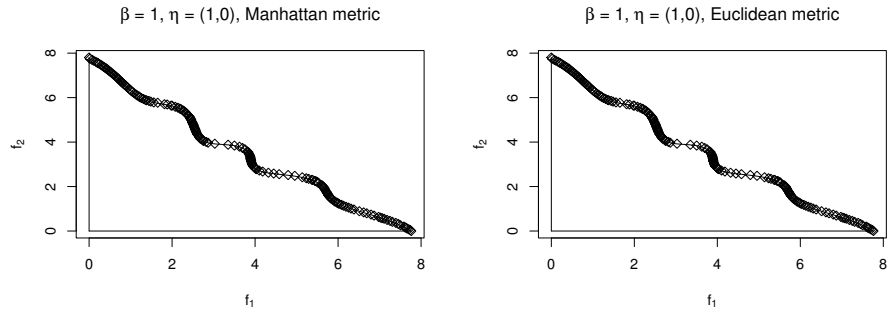


(a) Euclidean metric with a β value of 10.

(b) Euclidean metric with a β value of 20.

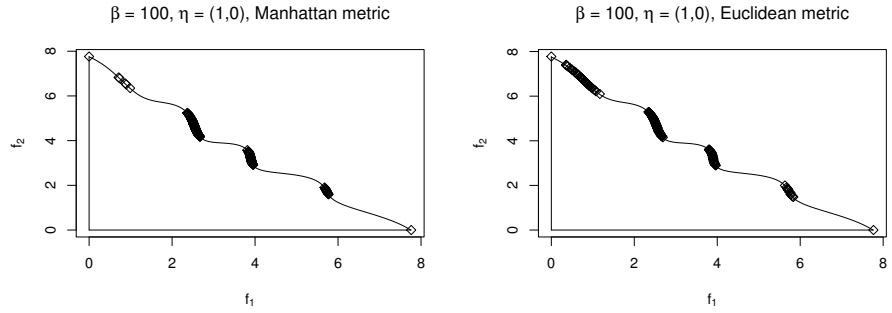
Figure 7.12: Pareto optimal front after 50 generations for projection based crowding applied to scenario 3 and using β values of 10 and 20.

Using $\vec{\eta} = (1, 0)$ does not result in anything unexpected as is evident from figure 7.13. The solutions along the Pareto optimal front are simply shifted toward one of the fitness functions.



(a) Manhattan metric for a β value of 1.

(b) Euclidean metric for a β value of 1.



(c) Manhattan metric for a β value of 100.

(d) Euclidean metric for a β value of 100.

Figure 7.13: Pareto optimal fronts after 50 generations for projection based crowding applied to scenario 3 using $\vec{\eta} = (1, 0)$.

Also, the results for the opposite case using $\vec{\eta} = (0, 1)$ are again so similar to those of figure 7.13 only shifted towards the other fitness function that they are not shown here.

This concludes the presentation of a rather extensive amount of results, when using the projection based crowding measure. Based upon these results, it is then possible to arrive at a conclusion.

7.4 Concluding Remarks

This chapter has been concerned with the projection based crowding scheme originally proposed in Branke and Deb (2004). Even before the crowding method was implemented a potential issue was identified which meant that the method was expected to produce poor results when putting an equal amount of emphasis on each of the fitness functions. A method to avoid this problem was proposed, such that the projection based crowding scheme would produce nice results independently of the emphasis put on each fitness function given by $\vec{\eta}$.

In total, 3 different scenarios were used to test the crowding measure, just as was the case in chapter 6 on page 105. Each of the scenarios had different shapes of the Pareto optimal front with certain properties that made it possible to get an idea of how well the projection based crowding method would perform.

The first thing that can be concluded is that, in order for the projection based crowding to be successful it had to use a Euclidean based distance metric instead of the Manhattan based metric normally used in NSGA-II. For all scenarios, the Manhattan based metric caused the method to be unsuccessful when the fitness functions were equally emphasized.

The amount of emphasis that was put on the difference between the original crowding distance and the crowding distance of the projected solutions also had quite an influence on the final distribution of solutions. Low values of β did not produce much change in the distribution, whereas a high value made it difficult in some cases to preserve certain areas of the Pareto optimal front. It was found that an intermediate value of β in the interval 10 to 20 produced distributions that not only emphasized on the desired curved regions, but also preserved a sparse distribution in the other regions as well, thus helping to preserve the overall structure of the Pareto optimal front. Since the method has only been tested on a small range of problems, it cannot be guaranteed that such nice distributions can be achieved for other problems as well. However, because the problems both included regions that were relatively flat and others that had a lot of curvature, it is expected that the method will work nicely for a wide range of other problems as well.

By changing the orientation of the hyperplane upon which the solutions were projected onto, it was possible to put the emphasis on different areas of the Pareto optimal front having different slope. This means that if some prior knowledge about a problem is known and if the different fitness functions should not be emphasized equally, then it can be used to shift the distributions to put emphasis on the desired regions as well.

In conclusion to this, it should be mentioned that the projection based crowding is one of the areas which will definitely help in automating the process of running MOEAs. By using a β value of 10 and putting an equal emphasis on all fitness functions, the method should produce very nice results with most emphasis in those regions where the largest tradeoffs are. The method will, however, still be able to maintain solutions along the entire Pareto optimal front, thus helping to give a better understanding of the problem being solved, since the shape of the entire Pareto optimal front is still distinguishable. The only drawback is that the calculation of the projected values are of an order $O(M^2)$, resulting in a total computational cost of $O(M^2 N \log N)$ for the algorithm. However, despite this increase in computational cost, it is still viable to use the method, since most problems usually keep the number of fitness functions at a relative low level around or below 10. A more pressing issue for the cases when M is large is that for each additional fitness function used a much higher population is needed in order to just create an adequate coverage.

As such, the computational cost will mostly be due to a large N for those situations rather than a large M.

With this comprehensive discussion of various distributions along the Pareto optimal front given in the last couple of chapters, the time has now come to take a look at ways of extending NSGA-II such that it will be possible to convert it more or less into a "black box" optimizer for any given problem.

Chapter 8

NSGA-II as an Optimization Tool

With the investigation of the crowding mechanism of NSGA-II all done, the time has now come to take a look at how the algorithm can be extended to the role of a "black box" optimization tool. Throughout the investigations performed in this thesis, the version of NSGA-II used has been the one which is freely available at `http://www.iitk.ac.in/kangal/`. The code of the algorithm is available as C code and as such, when the experiments for the previous chapters were conducted, they were compiled into the NSGA-II code before execution. However, using that method was not without flaws. For instance, when conducting the experiment using the toy problem in section 5.1 on page 84, the fitness functions were based on frequency responses, which required the use of complex numbers. As such, it was necessary to also incorporate complex numbers into the algorithm in order to be able to run the experiment. This comes to show that in order for the algorithm to be truly successful, it needs to either include all sorts of different specialized code or be able to run independently of the code which have such specialized requirements. This is what has led to the further development of the NSGA-II algorithm into a stand alone black box optimizer.

8.1 Splitting the Code

The key principle to creating the NSGA-II algorithm as a black box optimizer lies in the fact that the code used for the fitness calculations must be separated from the rest of the algorithm. By performing this split it is not necessary for the NSGA-II code to include anything other than the interface for the fitness calculation code. This interface can then be either based on message passing between processes, shared memory, or by using files. The disadvantage of using message passing and shared memory is that some specialized operating system specific code needs to be implemented in order for the method to work, thus limiting the use of the algorithm to certain systems.

The way the code of NSGA-II has been split in order to obtain the black box optimization structure proposed here is thus by using files. By allowing the algorithm to write the individuals to a file and then reading the calculated fitness of those same individuals from another file, it is possible to use the algorithm in connection with virtually any other program. By even allowing NSGA-II to read the input parameters from a file it is even possible to spawn the fitness calculation program from inside the NSGA-II algorithm itself by means of simple fork instructions.

In order to ensure that data are valid for both NSGA-II and the fitness calculation program a simple handshaking protocol is used, which is illustrated in the next couple of figures. The first details illustrated in figure 8.1 are connected to the initialization of the two programs.

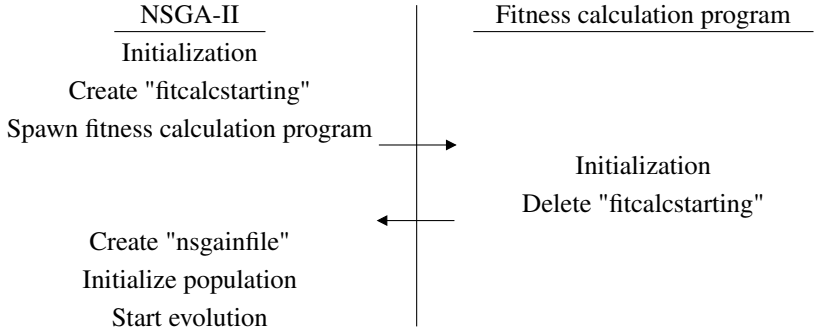


Figure 8.1: After the initialization of the NSGA-II optimization tool the fitness calculation program is spawned and NSGA-II waits for the deletion of a specific file before starting the evolutionary process.

The figure shows how NSGA-II can spawn the fitness calculation program and how the fitness calculation program can signal to NSGA-II that it is ready to receive input. This signal is comprised of the deletion of a specific file "fitcalcstarting", which is created by NSGA-II. When deleted by the fitness calculation server it is indicated to NSGA-II that initialization of the fitness calculation program is done and that the evolutionary process can start. The last action performed by NSGA-II before starting evolution is to create the file "nsgainfile" which is used for controlling the further flow. In figure 8.2 on the facing page, the flow during the evolutionary process is illustrated.

The key principle when it comes to the flow is the use of two distinct files which act as input/output files for NSGA-II and corresponding output/input files for the fitness calculation program. Each of the programs read data from their input files respectively, but only after the corresponding output file of that same program is deleted. As such, the deletion of the output file of a program acts as a signal to that program that the data contained in the input file is valid. This crude handshaking signal thus effectively controls the flow of the two programs making sure that data are always valid before they are read into each of the programs.

When it comes to termination of the evolutionary process, the principle is illustrated in figure 8.3 on the facing page.

By creating the file "fitcalcshutdown", it is signaled to the fitness termination program that it should terminate. NSGA-II then waits for the deletion of that same file which indicates that the

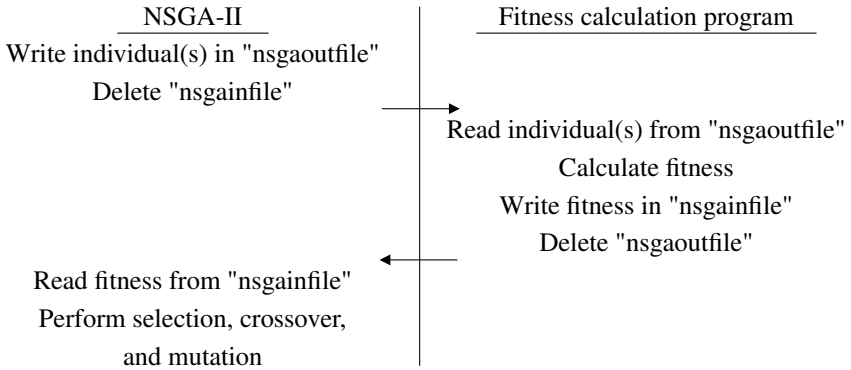


Figure 8.2: NSGA-II writes the individuals to a file "nsgaoutfile" and signals to the fitness calculation program that the data is valid by deleting the file "nsgainfile". After fitness calculation is done the fitness calculation program writes the result in a file "nsgainfile" and signals NSGA-II that data are valid by deleting the file "nsgaoutfile".

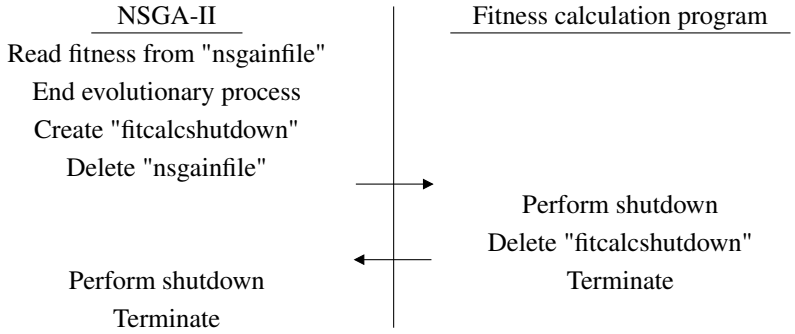


Figure 8.3: The shutdown process is started by the creation of the file "fitcalcshutdown" after which the fitness calculation program will perform the necessary steps to shut down and delete that same file indicating to NSGA-II that shutdown is successful.

fitness calculation program has been successfully shut down. After that, NSGA-II also terminates. The reason why NSGA-II is set to wait for the fitness calculation server to shut down is because of the possibility that a script might be used for running several consecutive instances of evolution. By waiting for the shutdown of the fitness calculation server, it is thus ensured that more than one instance of the fitness calculation programs does not overlap with any other instances that might be created during subsequent runs.

The only problem with this method arises if the evolution is interrupted prematurely or if one of the programs hangs, in which case the other program will not be aware of that fact and continues to wait. However, if this situation arises, it is quite simple to just terminate the programs manually and provided that the hangup was not due to program errors, it is possible to simply restart the algorithm again. If program errors did occur, it would be prudent to resolve the issue before restarting the algorithm since the problem might again occur and cause further hangups.

When writing the files used for communication between NSGA-II and the fitness calculation program, it is possible to vary the number of individuals that is written in them. In some cases, it might be desirable to minimize the wait and input/output times, thus writing the entire population in one file. In other cases, it might be more desirable to write each individual in a file of its own, which would be required if several instances of the fitness calculation program are to be run in parallel on different machines. However, the method has not been designed nor tested for running this black box version of NSGA-II on several machines in parallel and no guarantees can thus be given with regard to the effectiveness or performance in such a case.

Now, having split the code of NSGA-II into an algorithmic and a fitness calculation part, thus making the algorithm independent of the way fitness calculations are performed, it is time to discuss the parameters which is recommended to be used when running the algorithm.

8.2 Setting the Parameters

In the previous couple of chapters a series of experiments have been performed. In connection with those experiments different variables have been tested whereas other have remained constant. The reason why most of the parameters used have remained constant is because they generally work well on any given problem and also because the effect of only a few parameters was the focus of investigation. As such, it would be impossible to investigate the effect of certain parameters if some of the other parameters were changed as well.

One of the key issues discovered when performing the experiments is that the experiments using binary representation generally had worse performance for the problems investigated and also took longer to evolve. When faced with an unknown optimization problem, it is thus recommended to attempt to solve the problem first using a real valued representation. When doing so, the set of parameters in table 8.1 on the facing page is considered to generally perform well.

The number of fitness functions and the number of variables depend on the specific situation and as mentioned previously, the use of real valued variables is recommended due to the findings made in both chapters 5 on page 83 and 6 on page 105.

There are some of the recommended settings that are built into NSGA-II and as such they are also highly recommended to use. This covers for instance the selection operator built into NSGA-II which uses tournaments with a size of two. Also, NSGA-II only includes one type of crossover

Parameter description	Type	Value	Designation
Variables	Reals	-	-
Selection operator	Tournament	2	s_{size}
Crossover operator	SBX	10	-
Crossover probability	-	0.9	p_c
Mutation operator	Polynomial	50	-
Mutation probability	-	$\frac{1}{n_{reals}}$	p_m

Table 8.1: The set of parameters that are generally recommended when using NSGA-II for solving different problems.

and mutation operators, but for these it is possible to change some parameters. Usually, the default parametric settings of 10 for the crossover operator and 50 for the mutation operator can safely be used for running experiments. However, when it comes to the probabilities for those parameters some considerations must be made. For the mutation operator a mutation probability of $\frac{1}{n_{reals}}$ where n_{reals} is the number of variables used, is usually recommended since it on average will correspond to performing a single mutation operation for each individual. Values higher than that might cause the algorithm to diverge and values below could result in premature convergence due to too little diversity being added. With regard to the crossover probability this is usually desired to be set rather high, such that the mixing of genetic material is largely encouraged. However, this setting of the crossover probability also depends on the problem difficulty. In some instances for very difficult problems, it might be necessary to raise this probability, but it is generally not recommended since it could very easily lead to premature convergence. In general, a recommendation based on several distinct issues are put forth in Goldberg (2002) for obtaining the most efficient evolutionary process, or finding the "sweet spot". The issues influencing this setting are the crossover probability, the selection pressure, and problem difficulty.

When it comes to determining the population size N, things get a bit more tricky. Some investigation into population sizing for MOEAs has previously been done in Khan (2002), but the issue still depends highly on the problem, especially the number of fitness functions used since the Pareto optimal set can increase drastically for each fitness function that is used. Also, if the fitness calculations take a long time to compute, it is not optimal to choose a very large population, but the population must still be large enough to ensure that the algorithm does not converge prematurely. Some publications regarding the subject of parameterless algorithms include methods for algorithms to automatically determine the best population size, but they generally require an additional amount of computation since that process is incremental (Lobo & Goldberg, 2004; Lima & Lobo, 2004; Pelikan & Lin, 2004). In some cases it can even be necessary to use fitness inheritance similar to that seen in Pelikan and Sastry (2004), but currently this field of research has not yet been applied to MOEAs.

A generic setting for the maximum number of generations τ_{EA} cannot be given, since it will always depend highly in the problem. Setting this value too low will severely limit the quality of the obtained results and a value too high will be a waste of computations. It is always better to choose this value a little higher than what is expected, since it is always possible to stop the evolution prematurely if the population has converged.

Finally, it is time to give recommendations with regard to the issues investigated in this thesis. As seen in chapter 5 on page 83, it is recommended to use a locally based scaling method emphasizing each fitness function equally, such that any unspecified bias towards any fitness function can be avoided. However, in order to get a more useful distribution of points, it is recommended to also use the projection based crowding method proposed in chapter 7 on page 127. Because of the issues encountered when using the original Manhattan based crowding metric, it is recommended to use a Euclidean based metric when using this crowding method. Further, for the projection based crowding method it is recommended to use an $\vec{\eta}$ that emphasizes each fitness function equally unless another distribution is desired. Also, a β value of 10 is highly recommended.

Having given this long list of recommendations, and if using the modified version of the NSGA-II algorithm that splits the algorithm from the fitness calculations, it should be possible to solve a very wide range of problems with fitness functions given in a variety of ways, maybe even using different computational tools such as MATLAB or R just to name a few. Let us finish this chapter with a few concluding remarks.

8.3 Concluding Remarks

In this chapter, a method was presented that allowed for the NSGA-II algorithm to come one step closer at becoming a true black box optimization algorithm. By splitting the algorithm from the fitness calculations, a large barrier that limited different ways of implementing the fitness functions was removed, thus providing more choices when it comes to the calculation of the fitness functions that are to be optimized.

Following the proposed method of separating fitness calculations from the algorithm, a set of recommended parameters to use in connection with NSGA-II was proposed. This discussion of the recommended parameters also included the need for using some of the methods proposed in chapter 5 on page 83 and chapter 7 on page 127 for obtaining the best possible set of solutions on the Pareto optimal front. With this information, it should thus be possible for most people with little or no insight in the field of evolutionary computation to use NSGA-II for optimization of a wide set of different problems.

The goal of being capable of automatically designing controllers using MOEAs is thus one step closer to being a reality. With this statement, it is now time for the final conclusion in order to wrap up the contents presented in this thesis.

Chapter 9

Conclusion

It is now time to sum up on the different issues discussed throughout this thesis. The whole thing started out in chapter 2 on page 5 with a comprehensive, yet brief introduction to the field of evolutionary computation meant for those who do not already have an insight of the subject. This included a mathematical description which also covered the use of MOEAs. Following that, a survey in chapter 3 on page 55 of previous work that attempted to use evolutionary computation for control purposes was given. After that in chapter 4 on page 73, a discussion of objectives and constraints were given where the different possibilities of implementing those aspects into fitness functions in evolutionary algorithms were addressed. The focus then shifted towards the use of MOEAs, especially the NSGA-II algorithm. The crowding mechanism of NSGA-II was extensively investigated in an attempt to make the distribution of the solutions on the Pareto optimal front as optimal as possible for different types of problems. In chapter 5 on page 83, it was the ability of NSGA-II to handle disparate scalings of the fitness functions, without putting bias on any of the fitness functions when it was not explicitly told to do so that was addressed. After that particular goal was achieved it was investigated in chapter 6 on page 105 how emphasis in curvature could be achieved for a two-dimensional case, and in chapter 7 on page 127 it was extended to an M-dimensional case as well. Having given a short summary of the work included in the thesis so far, let us take a look at some of the main conclusions that can be drawn.

9.1 Main Conclusions

The main conclusions that can be drawn on the research performed in this thesis are as follows:

- It was successfully attempted to extend the general EA framework of Bäck (1996) to also cover the field of MOEAs as well. Where the original framework had only encompassed the single-objective algorithm, the extension of the resulting set allows for a variety of different MOEAs to be covered by the extended version as well.
- The survey showed that an extensive use of evolutionary computation within the field of control engineering has already taken place. Unfortunately, some of the theoretical foun-

dation for those uses have been lacking, since it is only within recent years that a solid theoretical framework for different areas of evolutionary computation has taken place.

- The issue that a biased distribution of solutions could result when the used fitness functions had disparate scalings was addressed and successfully resolved using a normalization method that was applied front-wise to the set of possible solutions.
- An investigation into different methods of obtaining a varied set of Pareto optimal solutions for different problems were performed. Some of the methods were only applicable to two dimensions, but the method based on projections was also applicable to higher dimensions as well. For the two dimensional case, a hybrid version of the original NSGA-II crowding and a derivative based approach was capable of achieving the desired distribution of points, but the projection based method also proved successful at achieving the desired distribution of solutions along the Pareto optimal front and is applicable to problems of any dimension.
- It was possible to separate the calculation of fitness functions from the main NSGA-II algorithm, such that it is now possible to use NSGA-II to solve a wide span of different problems that require specific programs in order to be calculated.
- Based on the default settings of NSGA-II, it was possible to find a default setting for the proposed projection based crowding such that the version of NSGA-II with the fitness calculations separated now can be used more or less as a black box optimizer.

Those were the feats achieved throughout the scope of this thesis. There are, however, still a few issues that were not investigated in detail and they remain open for further research.

9.2 Future Work

In the short run, the issues that need to be further investigated are those concerning the priorities of fitness functions. It was mentioned in chapter 4 on page 73 what the differences were when implementing constraints either as regular fitness functions or as specific constraints. It is thus a question what the effect will be if several layers of fitness functions with different priorities were to be combined and whether it would result in a more efficient search or not.

Another issue that would be interesting to investigate is if or how it might be possible to apply the discrete Gauss-Bonnet theorem to the set of Pareto optimal solutions. It is expected that an investigation into this area will give a better understanding of why the projection based crowding measure was capable of finding the desired distribution of solutions and if there might even be other and more suitable methods of achieving this goal.

In the long run, it is hoped that the research presented in this thesis will form the foundation upon which it might be possible to fully automate the design of controllers for various purposes. This will, among other things, include issues such as using GP for representation and also some simple guidelines for formulating the different fitness functions.

Also, the possibilities of extending the work in automatic controller design to cover efficient online approaches for finding the optimal controllers are worth investigating, since the ever expanding array of systems tend to become more and more autonomous all the time. If those systems would be able to evolve their own controllers, it would be a major breakthrough in the way systems are designed and will open up for a wide range of new possibilities.

The last issue that would be interesting to investigate further is an effective combination of the fields of evolutionary computation, neural networks, and fuzzy logic. The reason why this is interesting is because those things are what make up the most successful thing created by nature, namely human beings. By combining these areas and thus reap the benefits of combining the blueprints contained in the genes, the neural networks who are used for controlling different motor skills, and the fuzzy logic that makes up reasoning at a higher level, it might be possible to create crude versions of artificial humans. However, the scope for achieving these feats is still far into the future.

Bibliography

Abu-Alola, A. H., & Gough, N. E. (1995). Identification and adaptive control of nonlinear processes using combined neural networks and genetic algorithms. See Pearson, Steele, and Albrecht (1995), pp. 396–399.

Agah, A. (1996). A genetic algorithm-based controller for decentralized multi-agent robotic systems. See IEEE (1996), pp. 431–436.

Alander, J. T. (1991). On finding the optimal genetic algorithms for robot control problems. In *Proceedings IROS '91 IEEE/RSJ International Workshop on Intelligent Robots and Systems '91*, Volume 3 (pp. 1313–1318). Osaka, Japan.

Alander, J. T. (1993). *Genetic algorithms and robot control, A review*. In Robotikdagar '93.

Alander, J. T., Moghadampour, G., & Törmänen, P. (1997). Evaluating the benefit of fuzzy logic for PID-control by means of genetic algorithms - case: frequency controller. In Alander, J. T. (Ed.), *Proceedings of the Third Nordic Workshop on Genetic Algorithms and their Applications* (pp. 321–332). Vaasa, Finland: Finnish Artificial Intelligence Society.

Alander, J. T., Ylinen, J., & Tyni, T. (1995). Elevator group control using distributed genetic algorithm. See Pearson, Steele, and Albrecht (1995), pp. 400–403.

Alba, E., Cotta, C., & Troya, J. M. (1996). Type-constrained genetic programming for rule-base definition in fuzzy logic controllers. See Koza, Goldberg, Fogel, and Riolo (1996), pp. 255–260.

Almássy, N., & Verschure, P. (1992). Optimizing self-organizing control architectures with genetic algorithms: The interaction between natural selection and ontogenesis. See Männer and Manderick (1992), pp. 451–460.

American Automatic Control Council (2003). *Proceedings of the American Control Conference*. Denver, CO, American Automatic Control Council: IEEE Service Center.

Angeline, P. J., Reynolds, R. G., McDonnell, J. R., & Eberhart, R. (Eds.) (1997). *Evolutionary Programming VI*. Indianapolis, IN: Springer-Verlag.

Åström, K. J., & Hägglund, T. (1995). *PID controllers: Theory, design, and tuning* (2nd ed.). Research Triangle Park, NC: Instrument Society of America.

Bäck, T. (1996). *Evolutionary algorithms in theory and practice*. Oxford University Press.

Bailey, J. E., & KrishnaKumar, K. (1987). Total energy control concepts applied to flight in windshear. (pp. 525–532). American Institute of Aeronautics and Astronautics, Monterey, CA: American Institute of Aeronautics and Astronautics.

Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.) (1999a). *Proceedings of the Genetic and Evolutionary Computation Conference 1999*, Volume 2. Orlando, FL: Morgan Kaufmann Publishers.

Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.) (1999b). *Proceedings of the Genetic and Evolutionary Computation Conference 1999*, Volume 1. Orlando, FL: Morgan Kaufmann Publishers.

Bentley, P. J., & Wakefield, J. P. (1998). Finding acceptable solutions in the pareto-optimal range using multiobjective genetic algorithms. In Chawdhry, P., Roy, R., & Pant, R. (Eds.), *Proceedings of the Second Online World Conference on Soft Computing in Engineering Design and Manufacturing (WSC2)*, Volume 5 (pp. 231–240). Springer-Verlag.

Biondo, S. J., & Drummond, C. J. (Eds.) (1994). *High-Tech Controls for Energy and Environment: Proceedings of the Adaptive Control Systems Technology Symposium*. Pittsburgh, PA: U.S. Department of Energy.

Bobbin, J., & Yao, X. (1997). Solving optimal control problems with a cost on changing control by evolutionary algorithms. See IEEE (1997), pp. 331–336.

Branke, J., & Deb, K. (2004). *Integrating user preference into evolutionary multi-objective optimization* (Technical Report 2004004). Kanpur, PIN 208016, India: Kanpur Genetic Algorithms Laboratory (KanGAL), Indian Institute of Technology Kanpur.

Branke, J., Deb, K., Dierolf, H., & Osswald, M. (2004). *Finding knees in multi-objective optimization* (Technical Report 2004010). Kanpur, PIN 208016, India: Kanpur Genetic Algorithms Laboratory (KanGAL), Indian Institute of Technology Kanpur.

Camacho, E. F. (Ed.) (2002). *Proceedings of the 15th IFAC World Congress*. Barcelona, Spain: Elsevier Science.

Caponetto, R., Fortuna, L., Muscato, G., & Xibilia, M. G. (1994). Genetic algorithms for controller order reduction. See IEEE (1994a), pp. 724–729.

Carse, B., & Fogarty, T. C. (1994). A new approach to genetics based machine learning in fuzzy controller design. Columbus, OH: IEEE Control Systems Society.

Carse, B., Fogarty, T. C., & Munro, A. (1995). Adaptive distributed routing using evolutionary fuzzy control. See Eshelman (1995), pp. 389–396.

Cartwright, H. M., & Tuson, A. L. (1994). Genetic algorithms and flowshop scheduling: Towards the development of a real-time process control system. See Fogarty (1994b), pp. 277–290.

Chellapilla, K. (1998). Automatic generation of nonlinear optimal control laws for broom balancing using evolutionary programming. See IEEE (1998), pp. 195–200.

Chen, B.-S., & Cheng, Y.-M. (1998). A structure-specified h_∞ optimal control design for practical applications: A genetic approach. *IEEE Transactions on Control Systems Technology*, *6*(6), 707–718.

Chen, K., Parmee, I. C., & Gane, C. R. (1997). A genetic algorithm for mixed-integer optimisation in power and water system design and control. In *Intelligent Hybrid Systems: Fuzzy Logic, Neural Networks, and Genetic Algorithms* (Chapter 13, pp. 311–330). Norwell, MA: Kluwer Academic Publishers.

Chen, Y., & Kawaji, S. (1999). Identification and control of nonlinear system using PIPE algorithm. See Furuhashi and Maeda (1999), pp. 60–65.

Chiaberge, M., Merelo, J. J., Reyneri, L. M., Prieto, A., & Zocca, L. (1994). A comparison of neural networks, linear controllers, genetic algorithms and simulated annealing for real time control. In Verleysen, M. (Ed.), *European Symposium on Artificial Neural Networks* (pp. 205–210). Brussels, Belgium: D-Facto public.

Chiou, J.-P., & Wang, F.-S. (1998). A hybrid method of differential evolution with application to optimal control problems of a bioprocess system. See IEEE (1998), pp. 627–632.

Cho, D.-J., & Gweon, D.-G. (1999). Intelligent control of VCR audio tuning process using a genetic estimator. *Proceedings of the Institution of Mechanical Engineers Part B*, *213*, 577–586.

Cliff, D., Harvey, I., & Husbands, P. (1993). General visual robot controller networks via artificial evolution. In Casasent, D. (Ed.), *Proceedings of the Society of Photo-optical Instrumentation Engineers Conference 1993 (SPIE93)*, Volume SPIE Conference Volume 2055 (pp. 271–282). San Diego, CA. Also available as University of Sussex School of Cognitive and Computing Sciences Technical Report CSRP318.

Coelho, J. P., de Moura Oliveira, P. B., & Cunha, J. B. (2002). Greenhouse air temperature control using the particle swarm optimisation algorithm. See Camacho (2002).

Coello Coello, C. A., Christiansen, A. D., & Aguirre, A. H. (1995). Multiobjective design optimization of counterweight balancing of a robot arm using genetic algorithms. In *Proceedings of the Seventh IEEE Conference on Tools with Artificial Intelligence* (pp. 20–23). Herndon, VA: IEEE Computer Society Press.

Coello Coello, C. A., Van Veldhuizen, D. A., & Lamont, G. B. (2002). *Evolutionary algorithms for solving multi-objective problems*. Genetic Algorithms and Evolutionary Computation. New York, NY: Kluwer Academic Publishers.

Conradie, A., Miikkulainen, R., & Aldrich, C. (2002). Adaptive control utilising neural swarming. In Langdon, W. B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A. C., Miller, J. F., Burke, E., & Jonoska, N. (Eds.), *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 60–67). New York, NY: Morgan Kaufmann Publishers.

Corne, D. W., Knowles, J. D., & Oates, M. J. (2000). The pareto envelope-based selection algorithm for multiobjective optimization. See Schoenauer, Deb, Rudolph, Yao, Lutton, Guervós, and Schwefel (2000), pp. 839–848.

Curtis, A. R. D. (1991). An application of genetic algorithms to active vibration control. *Journal of Intelligent Material Systems and Structures*, *2*, 472–481.

Dadios, E. P., & Williams, D. J. (1996). A fuzzy-genetic controller for the flexible pole-cart balancing problem. See IEEE (1996), pp. 223–228.

Dadios, E. P., & Williams, D. J. (1998). Nonconventional control of the flexible pole-cart balancing problem: Experimental results. *IEEE Transactions on Systems, Man, and Cybernetics*, *28*(6), 895–901.

Dai, J., & Mao, J. (2002). Robust flight controller design for helicopters based on genetic algorithms. See Camacho (2002).

Dakev, N. V., Chipperfield, A. J., & Fleming, P. J. (1996). An evolutionary approach for path following optimal control of multibody systems. See IEEE (1996), pp. 512–516.

Deb, K. (1999). Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary Computation*, *7*(3), 205–230.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms* (1st ed.). West Sussex, England: John Wiley & Sons, Ltd.

Deb, K. (2002). *Unveiling innovative design principles by means of multiple conflicting objectives* (Technical Report 2002007). Kanpur, PIN 208016, India: Kanpur Genetic Algorithms Laboratory (KanGAL), Indian Institute of Technology Kanpur.

Deb, K., & Jain, S. (2002). *Multi-speed gearbox design using multi-objective evolutionary algorithms* (Technical Report 2002001). Kanpur, PIN 208016, India: Kanpur Genetic Algorithms Laboratory (KanGAL), Indian Institute of Technology Kanpur.

Deb, K., Mohan, M., & Mishra, S. (2003). *A fast multi-objective evolutionary algorithm for finding well-spread pareto-optimal solutions* (Technical Report 2003002). Kanpur, PIN 208016, India: Kanpur Genetic Algorithms Laboratory (KanGAL), Indian Institute of Technology Kanpur.

Deb, K., Poli, R., Banzhaf, W., Beyer, H.-G., Burke, E., Darwen, P., Dasgupta, D., Floreano, D., Foster, J., Harman, M., Holland, O., Lanzi, P. L., Spector, L., Tettamanzi, A., Thierens, D., & Tyrrell, A. (Eds.) (2004). *Genetic and Evolutionary Computation – GECCO-2004, part II*, Volume 3103 of *Lecture Notes in Computer Science*. Seattle, WA: Springer-Verlag.

Deb, K., Pratap, A., & Moitra, S. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. See Schoenauer, Deb, Rudolph, Yao, Lutton, Guervós, and Schwefel (2000), pp. 849–858.

Dittrich, P., Skusa, A., Banzhaf, W., & Kantschik, W. (1999). Dynamical properties of the fitness landscape of a GP controlled random morphology robot. See Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiela, and Smith (1999a), pp. 1002–1008.

Donha, D. C., Desanj, D. S., & Katebi, M. R. (1997). Genetic algorithm for weight selection in h_∞ control design. In Bäck, T. (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms* (pp. 599–606). East Lansing, MI: Morgan Kaufmann Publishers.

Dorigo, M. (1995). ALECSYS and the AutonoMouse: Learning to control a real robot by distributed classifier systems. *Machine Learning Journal*, *19*(3), 209–240.

Dracopoulos, D. C. (1997). Evolutionary control of a satellite. See Koza, Kalyanmoy, Dorigo, Fogel, Garzon, Iba, and Riolo (1997), pp. 77–81.

Ebner, M., & Zell, A. (1999). Evolving a behavior-based control architecture - From simulations to the real world. See Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiela, and Smith (1999a), pp. 1009–1014.

Eshelman, L. J. (Ed.) (1995). *Proceedings of the Sixth International Conference on Genetic Algorithms*. Pittsburgh, PA: Morgan Kaufmann Publishers.

Fang, X., Kellogg, B., Conlan, T., Dickerson, J., & Cook, D. (2003). High dimensional system design using genetic algorithms & visualization. See American Automatic Control Council (2003), pp. 4561–4566.

Fleming, P. J., & Fonseca, C. M. (1993). Genetic algorithms in control systems engineering: A brief introduction. *IEE Colloquium on Genetic Algorithms for Control Systems Engineering*, *130*, 1/1–1/5.

Fogarty, T. C. (1994a). Co-evolving co-operative populations of rules in learning control systems. See Fogarty (1994b), pp. 195–209.

Fogarty, T. C. (Ed.) (1994b). *Evolutionary Computing: AISB Workshop 1994*. Leeds, UK.

Fogarty, T. C., & Huang, R. (1991). *Systems control with the genetic algorithm and nearest-neighbour classification*. The University of West England.

Fogel, D. B. (1991). *System identification through simulated evolution: A machine learning approach to modeling*. Needham Heights: Ginn Press.

Fogel, D. B. (1992). *Evolving artificial intelligence*. Doctoral dissertation, University of California, San Diego, CA.

Fogel, L. J. (1962). Autonomous automata. *Industrial Research*, *4*, 14–19.

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). *Artificial intelligence through simulated evolution*. New York, NY: John Wiley & Sons, Inc.

Fonseca, C. M. (1995). *Multiobjective genetic algorithms with application to control engineering problems*. Unpublished doctoral dissertation, The University of Sheffield, Dept. of Automatic Control and Systems Engineering, Sheffield, UK.

Fonseca, C. M., & Fleming, P. J. (1993a). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. See Forrest (1993), pp. 416–423.

Fonseca, C. M., & Fleming, P. J. (1993b). Multiobjective genetic algorithms. *IEE Colloquium on Genetic Algorithms for Control Systems Engineering*, *130*, 6/1–6/5.

Fonseca, C. M., & Fleming, P. J. (1994). Multiobjective optimal controller design with genetic algorithms. In *Proceedings of the IEE Control '94 International Conference*, Volume 1 (pp. 745–749). Coventry, UK: IEE.

Forrest, S. (Ed.) (1993). *Proceedings of the Fifth International Conference on Genetic Algorithms*. Urbana-Champaign, IL: Morgan Kaufmann Publishers.

Forrest, S., & Perelson, A. S. (1991). Genetic algorithms and the immune system. See Schwefel and Männer (1991), pp. 320–325.

Forrest, S., Smith, R. E., Javornik, B., & Perelson, A. S. (1993). Using genetic algorithms to explore pattern recognition in the immune system. *Evolutionary Computation*, *1*(3), 191–211.

Freisleben, B., & Strelen, S. (1995). A hybrid genetic algorithm/fuzzy logic approach to manufacturing process control. See IEEE (1995a), pp. 837–841.

French, R. L. B., & Damper, R. I. (2001). Evolving a nervous system of spiking neurons for a behaving robot. See Spector, Goodman, Wu, Langdon, Voigt, Gen, Sen, Dorigo, Pezeshk, Garzon, and Burke (2001), pp. 1099–1106.

Fukunaga, A., Marks, J., & Ngo, J. T. (1994). Automatic control of physically realistic animated figures using evolutionary programming. See Sebald and Fogel (1994), pp. 76–83.

Fukuyama, Y., Takayama, S., Nakanishi, Y., & Yoshida, H. (1999). A particle swarm optimization for reactive power and voltage control in electric power systems. See Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiela, and Smith (1999a), pp. 1523–1528.

Furuhashi, T., & Maeda, J. (Eds.) (1999). *Proceedings of International Workshop on Soft Computing in Industry 1999*. Muroran, Hokkaido, Japan: IEEE Industrial Electronics Society.

Gilbert, A. H., Bell, F., & Valenzuela, C. L. (1995). Adaptive learning of process control and profit optimization using a classifier system. *Evolutionary Computation*, *3*(2), 177–198.

Gill, M. A. C., & Zomaya, A. Y. (1995). Genetic algorithms for robot control. See IEEE (1995b), pp. 462–466.

Goldberg, D. E. (1985a). Controlling dynamic systems with genetic algorithms and rule learning. In *Proceedings of The Fourth Yale Workshop on Applications of Adaptive Systems Theory* (pp. 91–97). Center for Systems Science, Becton Center, Yale University.

Goldberg, D. E. (1985b). Dynamic system control using rule learning and genetic algorithms. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, Volume 1 (pp. 588–592). Los Angeles, CA: Morgan Kaufmann Publishers.

Goldberg, D. E. (1985c). Genetic algorithms and rule learning in dynamic system control. In Grefenstette, J. J. (Ed.), *Proceedings of an International Conference on Genetic Algorithms and Their Applications* (pp. 8–15). Pittsburgh, PA: Lawrence Erlbaum Associates.

Goldberg, D. E. (1987). Computer-aided pipeline operation using genetic algorithms and rule learning. Part II: Rule learning control of a pipeline under normal and abnormal conditions. *Engineering with Computers*, *3*, 47–58.

Goldberg, D. E. (1988). Genetic algorithms in adaptive control: Why bother? In *Proceedings of the International Workshop on Adaptive Strategies for Industrial Use* (pp. 283–293). Lodge Kananaskis, AB.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.

Goldberg, D. E. (2002). *The design of innovation: Lessons from and for competent genetic algorithms*. Genetic Algorithms and Evolutionary Computation. Norwell, MA: Kluwer Academic Publishers.

Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, *3*(5), 493–530. (Also IlliGAL TCGA Report 89003).

Goldberg, D. E., Sastry, K., & Latoza, T. (2001). On the supply of building blocks. See Spector, Goodman, Wu, Langdon, Voigt, Gen, Sen, Dorigo, Pezeshk, Garzon, and Burke (2001), pp. 336–342.

Goldberg, D. E., & Segrest, P. (1987). Finite Markov chain analysis of genetic algorithms. In Grefenstette, J. J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 1–8). Cambridge, MA: Lawrence Erlbaum Associates.

Goldberg, D. E., & Wang, L. (1997). Adaptive niching via coevolutionary sharing. See Quagliarella, Périaux, Poloni, and Winter (1997) (Chapter 2, pp. 21–38).

Grant, P. R., & Grant, B. R. (2002). Adaptive radiation of darwin's finches. *American Scientist*, *90*(2), 130.

Grefenstette, J. J. (1989a). Incremental learning of control strategies with genetic algorithms. In Spatz, B. (Ed.), *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 340–344). Ithaca, NY: Morgan Kaufmann Publishers.

Grefenstette, J. J. (1989b). A system for learning control strategies with genetic algorithms. See Schaffer (1989), pp. 183–190.

Gritz, L., & Hahn, J. K. (1997). Genetic programming evolution of controllers for 3-D character animation. See Koza, Kalyanmoy, Dorigo, Fogel, Garzon, Iba, and Riolo (1997), pp. 139–146.

Harik, G. R., & Goldberg, D. E. (1996). Learning linkage. In *Foundations of Genetic Algorithms IV* (pp. 247–262). San Mateo, CA: Morgan Kaufmann Publishers.

Harvey, I. (1992). Species adaptation genetic algorithms: A basis for a continuing SAGA. In Varela, F. J., & Bourgine, P. (Eds.), *Toward a Practice of Autonomous Systems: Proceed-*

ings of the First European Conference on Artificial Life (pp. 346–354). Paris, France: MIT Press.

Harvey, I., Husbands, P., & Cliff, D. (1993). *Genetic convergence in a species of evolved robot control architectures*. (Research Report No. CSRP 267, University of Sussex, School of Cognitive and Computing Sciences, UK).

Herrera, F., Lozano, M., & Verdegay, J. L. (1998). A learning process for fuzzy control rules using genetic algorithms. *Fuzzy Sets and Systems*, *100*, 143–158.

Herrera, F., & Verdegay, J. L. (Eds.) (1996). *Genetic Algorithms and Soft Computing*, Volume 8 of *Studies in Fuzziness and Soft Computing*. Heidelberg, Germany: Physica-Verlag.

Hightower, R. R., Forrest, S., & Perelson, A. S. (1995). The evolution of emergent organization in immune system gene libraries. See Eshelman (1995), pp. 344–350.

Hoffman, F. (1997). Evolutionary learning of mobile robot behaviors. In Wang, P. P. (Ed.), *Proceedings of the Joint Conference on Information Sciences: Fuzzy Logic, Intelligent Control & Genetic Algorithm*, Volume 1 (pp. 104–107). Research Triangle Park, NC: Duke University.

Hoffmann, F., & Pfister, G. (1996). Learning of a fuzzy control rule base using messy genetic algorithms. See Herrera and Verdegay (1996) (pp. 279–305).

Hofmeyr, S. A., & Forrest, S. (1999). Immunity by design: An artificial immune system. See Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiela, and Smith (1999a), pp. 1289–1296.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Hollstein, R. B. (1971). *Artificial genetic adaptation in computer control systems*. Doctoral dissertation, University of Michigan. (University Microfilms No. 71-23,773).

Hondo, N., Nishikawa, K., Yokoi, H., & Kakazu, Y. (1998). Multi-agent programming system for starfish robot control. See Koza, Banzhaf, Chellapilla, Deb, Dorigo, Fogel, Garzon, Goldberg, Iba, and Riolo (1998), pp. 140–145.

Hornby, G. S., Takamura, S., Hanagata, O., Fujita, M., & Pollack, J. (2000). Evolution of controllers from a high-level simulator to a high DOF robot. In Miller, J., Thompson, A., Thomson, P., & Fogarty, T. C. (Eds.), *Third International Conference on Evolvable Systems: From Biology to Hardware* (pp. 80–89). Edinburgh, Scotland: Springer-Verlag.

Howley, B. (1996). Genetic programming of near-minimum-time spacecraft attitude maneuvers. See Koza, Goldberg, Fogel, and Riolo (1996), pp. 98–106.

Howley, B. (1997). Genetic programming and parametric sensitivity: a case study in dynamic control of a two link manipulator. See Koza, Kalyanmoy, Dorigo, Fogel, Garzon, Iba, and Riolo (1997), pp. 180–185.

Human Genome Project (2004). Human genome project. Website. `http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml`.

Hunt, K. J. (1992a). Optimal control system synthesis with genetic algorithms. See Männer and Manderick (1992), pp. 381–389.

Hunt, K. J. (1992b). Polynomial lqg and H_∞ controller synthesis: A genetic algorithm solution. In *Proceedings of the 31st IEEE Conference on Decision and Control* (pp. 3604–3609). IEEE, Tucson, AZ: IEEE Press.

Hwang, W.-R., & Zein-Sabatto, S. (1997). Fuzzy controller design using genetic algorithms. In *Proceedings of IEEE Southeastcon '97* (pp. 6–10). IEEE, Blacksburg, VA: IEEE Service Center.

IEEE (1994a). *Proceedings of the First IEEE Conference on Evolutionary Computation*, Volume 2. Orlando, FL, IEEE: IEEE Service Center.

IEEE (1994b). *Proceedings of the First IEEE Conference on Evolutionary Computation*, Volume 1. Orlando, FL, IEEE: IEEE Service Center.

IEEE (1995a). *Proceedings of 1995 IEEE International Conference on Evolutionary Computation*, Volume 2. Perth, WA, Australia, IEEE: IEEE Service Center.

IEEE (1995b). *Proceedings of 1995 IEEE International Conference on Evolutionary Computation*, Volume 1. Perth, WA, Australia, IEEE: IEEE Service Center.

IEEE (1996). *Proceedings of 1996 IEEE International Conference on Evolutionary Computation*. Nagoya, Japan, IEEE: IEEE Service Center.

IEEE (1997). *Proceedings of 1997 IEEE International Conference on Evolutionary Computation*. Indianapolis, IN, IEEE: IEEE Service Center.

IEEE (1998). *Proceedings of 1998 IEEE International Conference on Evolutionary Computation*. Anchorage, AK, IEEE: IEEE Service Center.

Ishida, Y., & Adachi, N. (1996). Active noise control by an immune algorithm: Adaptation in immune system as an evolution. See IEEE (1996), pp. 150–154.

Iwasaki, M., Miwa, M., & Matsui, N. (1999). Autonomous identification of motion control system using genetic algorithms. See Furuhashi and Maeda (1999), pp. 377–382.

Jamshidi, M., dos Santos Coelho, L., Krohling, R. A., & Fleming, P. J. (2002). *Robust control systems with genetic algorithms*, Volume 3 of *Control Series*. CRC Press.

Johnson, G. (2004). The evidence for evolution. Website. `http://www.txtwriter.com/Backgrounders/Evolution/EVcontents.html`.

Jones, A. H. (1995). Genetic tuning of neural non-linear PID controllers. See Pearson, Steele, and Albrecht (1995), pp. 412–415.

Kacprzyk, J. (1995). Multistage control of a fuzzy system using a genetic algorithm. See IEEE (1995a), pp. 842–845.

Kacprzyk, J. (1996). Genetic algorithms in multistage fuzzy control. See Herrera and Verdegay (1996) (pp. 579–598).

Karr, C. L. (1991). Design of an adaptive fuzzy logic controller using a genetic algorithm. In Belew, R. K., & Booker, L. B. (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 450–457). San Diego, CA: Morgan Kaufmann Publishers.

Karr, C. L. (1992). An adaptive system for process control using genetic algorithms. In Verbruggen, H. B., & Rodd, M. G. (Eds.), *Artificial Intelligence in Real-Time Control 1992* (pp. 329–334). Oxford, UK: Pergamon Press.

Karr, C. L., & Gentry, E. J. (1993). Fuzzy control of pH using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, *1*(1), 46–53.

Karr, C. L., Meredith, D. L., & Stanley, D. A. (1990). Fuzzy process control with a genetic algorithm. *Control '90 – Mineral and Metallurgical Processing*, 53–60.

Kawaji, S., Ogasawara, K., & Honda, H. (1994). Swing up control of a pendulum using genetic algorithms. In Peshkin, M. (Ed.), *Proceedings of the 33rd IEEE Conference on Decision and Control*, Volume 4 (pp. 3530–3532). Piscataway, NJ: IEEE Service Center.

Khan, N. (2002). *Bayesian optimization algorithms for multiobjective and hierarchically difficult problems*. Master's thesis, University of Illinois at Urbana-Champaign, Urbana, IL. Also available as IlliGAL Report No. 2003021.

Kim, H.-S., Kim, J.-H., & Choi, Y.-K. (1996). Variable structure control of brushless DC motor using evolution strategy with varying search space. See IEEE (1996), pp. 764–769.

Kim, J.-H., & Jeon, J.-Y. (1996). Evolutionary programming-based high-precision controller design. In Fogel, L., Angeline, P., & Bäck, T. (Eds.), *Evolutionary Programming V* (pp. 73–81). San Diego, CA: MIT Press.

Kim, J.-H., & Moon, B.-R. (2001). Adaptive and dynamic elevator group control with a genetic algorithm. See Spector, Goodman, Wu, Langdon, Voigt, Gen, Sen, Dorigo, Pezeshk, Garzon, and Burke (2001), pp. 1322–1329.

Kim, J.-H., & Shim, H.-S. (1995). Evolutionary programming-based optimal robust locomotion control of autonomous mobile robots. See McDonnell, Reynolds, and Fogel (1995), pp. 631–644.

Kinzel, J., Klawonn, F., & Kruse, R. (1994). Modifications of genetic algorithms for designing and optimizing fuzzy controllers. See IEEE (1994b), pp. 28–33.

Knowles, J. D., & Corne, D. W. (1999). The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Proceedings of the 1999 Congress on Evolutionary Computation* (pp. 98–105). IEEE, Washington DC: IEEE Service Center.

Koza, J. R. (1991). Evolution and co-evolution of computer programs to control independently-acting agents. In Meyer, J.-A., & Wilson, S. W. (Eds.), *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior (SAB-90)* (pp. 366–375). Paris, France: MIT Press.

Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H., & Riolo, R. L. (Eds.) (1998). *Genetic Programming 98*. Madison, WI: Morgan Kaufmann Publishers.

Koza, J. R., Bennett III, F. H., Keane, M. A., & Andre, D. (1997). Evolution of a time-optimal fly-to controller circuit using genetic programming. See Koza, Kalyanmoy, Dorigo, Fogel, Garzon, Iba, and Riolo (1997), pp. 207–212.

Koza, J. R., Goldberg, D. E., Fogel, D. B., & Riolo, R. L. (Eds.) (1996). *Genetic Programming 1996*. Stanford, CA: MIT Press.

Koza, J. R., Kalyanmoy, D., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H., & Riolo, R. L. (Eds.) (1997). *Genetic Programming 1997*. Stanford, CA: Morgan Kaufmann Publishers.

Koza, J. R., & Keane, M. A. (1990, January). Cart centering and broom balancing by genetically breeding populations of control strategy programs. In Caudill, M. (Ed.), *Proceedings of the International Joint Conference on Neural Networks* (pp. 198–201). Washington DC: Lawrence Erlbaum Associates.

Koza, J. R., Keane, M. A., Yu, J., Bennett III, F. H., & Mydlowec, W. (2000). Automatic creation of human-competitive programs and controllers by means of genetic programming. *Genetic Programming and Evolvable Machines*, *1*(1/2), 121–164.

Koza, J. R., Yu, J., Keane, M. A., & Mydlowec, W. (2000). Evolution of a controller with a free variable using genetic programming. In Poli, R., Banzhaf, W., Langdon, W. B., Miller, J., Nordin, P., & Fogarty, T. C. (Eds.), *Genetic Programming, EuroGP 2000* (pp. 91–105). Edinburgh, Scotland, UK: Springer-Verlag.

Krink, T., Ursem, R. K., & Filipic, B. (2001). Evolutionary algorithms in control optimization: The greenhouse problem. See Spector, Goodman, Wu, Langdon, Voigt, Gen, Sen, Dorigo, Pezeshk, Garzon, and Burke (2001), pp. 440–447.

Krishna, K., & Naik, V. K. (2000). Application of evolutionary algorithms in controlling semi-autonomous mission-critical distributed systems. See Wang, Chen, Cheng, Gattiker, Georgiou, Laws, Lu, Romay, Scwartz, Shenoi, Shih, Wang, and Ying (2000), pp. 1015–1018.

KrishnaKumar, K., & Goldberg, D. E. (1990). Genetic algorithms in control system optimization. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference* (pp. 1568–1577). American Institute of Aeronautics and Astronautics, Portland, OR: American Institute of Aeronautics and Astronautics.

KrishnaKumar, K., & Goldberg, D. E. (1992). Control system optimization using genetic algorithms. *Journal of Guidance, Control, and Dynamics*, *15*(3), 735–740.

KrishnaKumar, K., & Satyadas, A. (1995). Evolving multiple fuzzy models and its application to an aircraft control problem. See Winter, Périaux, Galán, and Cuesta (1995) (Chapter 16, pp. 305–320).

KrishnaKumar, K. S. (1988). *Energy concepts applied to control of airplane flight in wind shear*. Unpublished doctoral dissertation, The University of Alabama, College of Engineering, Tuscaloosa, AL.

Kristinsson, K. (1989). *Genetic algorithms in system identification and control*. Unpublished master's thesis, University of British Columbia, Department of Electrical Engineering, Vancouver, Canada.

Kristinsson, K., & Dumont, G. A. (1991). *System identification and control using genetic algorithms* (Tech. Rep. No. PGRL483). Vancouver, Canada: University of British Columbia, Department of Electrical Engineering.

Kuchinski, M. J. (1985). *Battle management systems control rule optimization using artificial intelligence* (Tech. Rep. No. NSWC MP 84-329). Dahlgren, VA: Naval Surface Weapons Center.

Kundu, S., & Kawata, S. (1996). AI in control system design using a new paradigm for design representation. In *Proceedings of the Fourth International Conference on Artificial Intelligence in Design* (pp. 1–18). Stanford, CA: Kluwer Academic Publishers.

Kundu, S., Kawata, S., & Watanabe, A. (1995). Modeling emergent collective behavior for control systems. In Louis, S. J. (Ed.), *Fourth Golden West International Conference on Intelligent Systems* (pp. 86–90). International Society for Computers and their Applications, San Francisco, CA: ISCA.

Kundu, S., Kawata, S., & Watanabe, A. (1996). A multicriteria approach to control system design with genetic algorithm. In Gertler, J. J., Cruz, J. B., & Peshkin, M. (Eds.), *Proceedings of the 13th IFAC World Congress*, Volume D (pp. 315–320). San Francisco, CA: Elsevier Science.

Kwok, D. P., & Sheng, F. (1994). Genetic algorithm and simulated annealing for optimal robot arm PID control. See IEEE (1994a), pp. 707–713.

Kwon, Y. D., Won, J. M., & Lee, J. S. (1998). Control of mobile robot by using evolutionary fuzzy controller. See IEEE (1998), pp. 422–427.

Lai, L. L., & Ma, J. T. (1995). Power flow control in facts using evolutionary programming. See IEEE (1995b), pp. 109–113.

Langballe, A. S., & Pedersen, G. K. M. (2001, June). *Controller design using evolutionary algorithms*. Master's thesis, Aalborg University.

Law, D., & Miikkulainen, R. (1994). *Grounding robotic control with genetic neural networks* (Tech. Rep. No. AI94-223). Austin, Texas: The University of Texas at Austin.

Lay, M.-Y. (1994). Application of genetic programming in analyzing multiple steady states of dynamical systems. See IEEE (1994b), pp. 333–336b.

Lee, W.-P., Hallam, J., & Lund, H. H. (1996). A hybrid GP/GA approach for co-evolving controllers and robot bodies to achieve fitness-specified tasks. See IEEE (1996), pp. 384–389.

Li, G., Tsang, K. M., Rad, A. B., & Chow, K. M. (1999). Genetic fuzzy control for a motor position servo system. See Furuhashi and Maeda (1999), pp. 252–257.

Li, Y., Tan, K. C., & Gong, M. (1997). Global structure evolution and local parameter learning for control system model reductions. In *Evolutionary Algorithms in Engineering Applications* (pp. 345–360). Berlin, Germany: Springer-Verlag.

Lichtfuss, H. J. (1965). *Evolution eines rohrkrümmers*. Master's thesis, Technical University of Berlin.

Lima, C. F., & Lobo, F. G. (2004). Parameter-less optimization with the extended compact genetic algorithm and iterated local search. In Deb, K., Poli, R., Banzhaf, W., Beyer, H.-G., Burke, E., Darwen, P., Dasgupta, D., Floreano, D., Foster, J., Harman, M., Holland, O., Lanzi, P. L., Spector, L., Tettamanzi, A., Thierens, D., & Tyrrell, A. (Eds.), *Genetic and Evolutionary Computation – GECCO-2004, Part I*, Volume 3102 of *Lecture Notes in Computer Science* (pp. 1328–1339). Seattle, WA: Springer-Verlag.

Lin, L.-C., & Lin, Y.-J. (1998). Fuzzy-enhanced adaptive control for flexible drive system with friction using genetic algorithms. *Journal of Intelligent and Robotic Systems*, *23*, 379–405.

Lin, S.-C., & Chen, Y.-Y. (1995). On GA-based optimal fuzzy control. See IEEE (1995a), pp. 846–851.

Lobo, F. G., & Goldberg, D. E. (2004). The parameter-less genetic algorithm in practice. *Information Sciences*, *167*, 217–232.

Louis, S. J., & Li, G. (1997). Combining robot control strategies using genetic algorithms with memory. See Angeline, Reynolds, McDonnell, and Eberhart (1997), pp. 431–441.

Männer, R., & Manderick, B. (Eds.) (1992). *Parallel Problem Solving from Nature, 2*. Brussels, Belgium: Elsevier Science.

Marra, M. A., Boling, B. E., & Walcott, B. L. (1996). Stability analysis of genetic algorithm controllers. In *Proceedings of IEEE Southeastcon '96* (pp. 204–207). IEEE, Tampa, FL: IEEE Service Center.

Mautner, C., & Belew, R. K. (1999). Coupling morphology and control in a simulated robot. See Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiela, and Smith (1999a), pp. 1350–1357.

McDonnell, J. R. (1997). Evolutionary computation applications: control. In Bäck, T., Fogel, D. B., & Michalewicz, Z. (Eds.), *Handbook of Evolutionary Computation* (pp. F1.3:1–F1.3:7). Bristol and New York: Institute of Physics Publishing and Oxford University Press.

McDonnell, J. R., Reynolds, R. G., & Fogel, D. B. (Eds.) (1995). *Evolutionary Programming IV*. San Diego, CA: MIT Press.

McGregor, D. R., Odetayo, M. O., & Dasgupta, D. (1992). Adaptive control of a dynamic system using genetic-based methods. In *Proceedings of the IEEE International Symposium on Intelligent Control* (pp. 521–525). Glasgow, Scotland: IEEE Control Systems Society.

Michalewicz, Z., Janikow, C. Z., & Krawczyk, J. B. (1992). A modified genetic algorithm for optimal control problems. *Computers & Mathematics with Applications*, *23*(12), 83–94.

Michalewicz, Z., Krawczyk, J. B., Kazemi, M., & Janikow, C. Z. (1990). Genetic algorithms and optimal control problems. In deCarlo, R. A. (Ed.), *Proceedings of the 29th IEEE Conference on Decision and Control* (pp. 1664–1666). Honolulu, HI: IEEE Press.

Mikami, S., Mitsuo, W., & Kakazu, Y. (1996). Combining reinforcement learning with GA to find co-ordinated control rules for multi-agent systems. See IEEE (1996), pp. 356–361.

Moin, N. H., Zinober, A. S. I., & Harley, P. J. (1995). Application of genetic algorithms in sliding mode control design. See Pearson, Steele, and Albrecht (1995), pp. 452–455.

Morimoto, T., Baerdemaeker, J. D., & Hashimoto, Y. (1997). An intelligent approach for optimal control of fruit-storage process using neural networks and genetic algorithms. *Computers and Electronics in Agriculture*, *18*, 205–224.

Neubauer, A. (1997). Genetic design of analog IIR filters with variable time delays for optically controlled microwave signal processors. See IEEE (1997), pp. 437–442.

Obayashi, S. (1997). Pareto genetic algorithm for aerodynamic design using the Navier-Stokes equations. See Quagliarella, Périaux, Poloni, and Winter (1997) (Chapter 12, pp. 245–266).

Odetayo, M. O., & McGregor, D. R. (1989). Genetic algorithm for inducing control rules for a dynamic system. See Schaffer (1989), pp. 177–182.

Ohno, H., & Furuhashi, T. (1999). A conceptual framework for construction of autonomous system and its application to input variable selection for cart pole control. See Furuhashi and Maeda (1999), pp. 48–53.

Oprea, M., & Forrest, S. (1999). How the immune system generates diversity: Pathogen space coverage with random and evolved antibody libraries. See Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiela, and Smith (1999a), pp. 1651–1656.

Ortega, G., & Giron-Sierra, J. M. (1995). Genetic algorithms for fuzzy control of automatic docking with a space station. See IEEE (1995b), pp. 157–161.

Park, J.-H., & Choi, Y.-K. (1996). An on-line PID control scheme for unknown nonlinear dynamic systems using evolution strategy. See IEEE (1996), pp. 759–763.

Park, Y. J., Cho, H. S., & Cha, D. H. (1995). Genetic algorithm-based optimization of fuzzy logic controller using characteristic parameters. See IEEE (1995a), pp. 831–836.

Parker, G. B. (2001). The incremental evolution of gaits for hexapod robots. See Spector, Goodman, Wu, Langdon, Voigt, Gen, Sen, Dorigo, Pezeshk, Garzon, and Burke (2001), pp. 1114–1121.

Parker, G. B., & Mills, J. W. (1999). Adaptive hexapod gait control using anytime learning with fitness biasing. See Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiela, and Smith (1999b), pp. 519–524.

Pearson, D. W., Steele, N. C., & Albrecht, R. F. (Eds.) (1995). *Proceedings of the 1995 International Conference on Artificial Neural Nets and Genetic Algorithms*. École des Mines d'Alès, France: Springer-Verlag.

Pedersen, G. K. M., & Goldberg, D. E. (2004). Dynamic uniform scaling for multiobjective genetic algorithms. See Deb, Poli, Banzhaf, Beyer, Burke, Darwen, Dasgupta, Floreano,

Foster, Harman, Holland, Lanzi, Spector, Tettamanzi, Thierens, and Tyrrell (2004), pp. 11–23.

Pedersen, G. K. M., Langballe, A. S., & Wiśniewski, R. (2002). Synthesizing multi-objective H_2/H_∞ dynamic controller using evolutionary algorithms. See Camacho (2002).

Pelikan, M., & Goldberg, D. E. (2001). Hierarchical bayesian optimization algorithm = bayesian optimization algorithm + niching + local structures. In *Optimization by Building and Using Probabilistic Models (OBUPM) 2001* (pp. 217–221). San Francisco, CA.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm. See Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiela, and Smith (1999b), pp. 525–532.

Pelikan, M., & Lin, T.-K. (2004). Parameter-less hierarchical BOA. See Deb, Poli, Banzhaf, Beyer, Burke, Darwen, Dasgupta, Floreano, Foster, Harman, Holland, Lanzi, Spector, Tettamanzi, Thierens, and Tyrrell (2004), pp. 24–35.

Pelikan, M., & Sastry, K. (2004). Fitness inheritance in the bayesian optimization algorithm. See Deb, Poli, Banzhaf, Beyer, Burke, Darwen, Dasgupta, Floreano, Foster, Harman, Holland, Lanzi, Spector, Tettamanzi, Thierens, and Tyrrell (2004), pp. 48–59.

Pham, D. T., & Karaboga, D. (1999). Self-tuning fuzzy controller design using genetic optimisation and neural network modelling. *Artificial Intelligence in Engineering*, *13*(2), 119–130.

Pipe, A. G., & Carse, B. (2000). Autonomous acquisition of fuzzy rules for mobile robot control: First results from two evolutionary computation approaches. See Whitley, Goldberg, Cantú-Paz, Spector, Parmee, and Beyer (2000), pp. 849–856.

Pohlheim, H., & Heißner, A. (1999). Optimal control of the greenhouse climate using real-world weather data and evolutionary algorithms. See Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiela, and Smith (1999a), pp. 1672–1677.

Porter, B., & Allaoui, C. (1995). Performance measures in the genetic design of digital controllers for robotic manipulators. See IEEE (1995a), pp. 509–514.

Porter II, L. M. L., & Passino, K. M. (1994). Genetic model reference adaptive control. Columbus, OH: IEEE Control Systems Society.

Pratt, P. (1994). Evolving neural networks to control unstable dynamical systems. See Sebald and Fogel (1994), pp. 191–204.

Quagliarella, D., Périaux, J., Poloni, C., & Winter, G. (Eds.) (1997). *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*. Trieste, Italy: John Wiley & Sons, Ltd.

Ram, A., Arkin, R., Boone, G., & Pearce, M. (1994). Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation. *Adaptive Behavior*, *2*(3), 277–304.

Rechenberg, I. (1995). *Cybernetic solution path of an experimental problem* (Technical Report). Farnborough, Hants., UK: Royal Aircraft Establishment. Library Translation No. 1122.

Renders, J. M., Nordvik, J. P., & Bersini, H. (1992). Genetic algorithms for process control: A survey. In Verbruggen, H. B., & Rodd, M. G. (Eds.), *Artificial Intelligence in Real-Time Control 1992* (pp. 323–328). Oxford, UK: Pergamon Press.

Rodrigues, L., Prado, M., Tavares, P., da Silva, K., & Rosa, A. (1996). Simulation and control of biped locomotion - GA optimization. See IEEE (1996), pp. 390–395.

Romzi, M., Nishino, J., Odaka, T., & Ogura, H. (1999). An acquisition of fuzzy control rules for inverted double pendulum system using genetic algorithm. See Furuhashi and Maeda (1999), pp. 264–269.

Ronald, E., & Schoenauer, M. (1994). Genetic lander: An experiment in accurate neuro-genetic control. In Davidor, Y., Schwefel, H.-P., & Männer, R. (Eds.), *Parallel Problem Solving from Nature - PPSN III*, Volume 866 of *Lecture Notes in Computer Science* (pp. 452–461). Jerusalem, Israel: Springer-Verlag.

Ronge, A., & Nordahl, M. G. (1996). Genetic programs and co-evolution: Developing robust general purpose controllers using local mating in 2-dimensional population. In Voigt, H.-M., Ebeling, W., Rechenberg, I., & Schwefel, H.-P. (Eds.), *Parallel Problem Solving from Nature - PPSN IV*, Volume 1141, of *Lecture Notes in Computer Science* (pp. 81–90). Berlin, Germany: Springer-Verlag.

Salomon, R. (1997). Scaling behavior of the evolution strategy when evolving neuronal control architectures for autonomous agents. See Angeline, Reynolds, McDonnell, and Eberhart (1997), pp. 47–57.

Sałustowicz, R., & Schmidhuber, J. (1997). Probabilistic incremental program evolution. *Evolutionary Computation*, *5*(2), 123–141.

Santos, J., Duro, R. J., Becerra, J. A., Crespo, J. L., & Bellas, F. (2000). Aspects of evolution for obtaining real robot controllers. See Wang, Chen, Cheng, Gattiker, Georgiou, Laws, Lu, Romay, Scwartz, Shenoi, Shih, Wang, and Ying (2000), pp. 1023–1026.

Saravanan, N. (1995). Evolutionary programming for synthesis of optimal controllers. See McDonnell, Reynolds, and Fogel (1995), pp. 645–656.

Saravanan, N., & Fogel, D. B. (1994). Evolving neurocontrollers using evolutionary programming. See IEEE (1994b), pp. 217–222.

Schaffer, J. D. (Ed.) (1989). *Proceedings of the Third International Conference on Genetic Algorithms*. Fairfax, VA: Morgan Kaufmann Publishers.

Schoen, M. P., Chinvorarat, S., & Schoen, G. M. (2003). Identification and robust control of flexible space structures. See American Automatic Control Council (2003), pp. 4585–4589.

Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Guervós, J. J. M., & Schwefel, H.-P. (Eds.) (2000). *Parallel Problem Solving from Nature - PPSN VI*, Volume 1917 of *Lecture Notes in Computer Science*. Paris, France: Springer-Verlag.

Schoenauer, M., & Ronald, E. (1994). Neuro-genetic truck backer-upper controller. See IEEE (1994a), pp. 720–723.

Schultz, A. C., Grefenstette, J. J., & De Jong, K. A. (1995). *Applying genetic algorithms to the testing of intelligent controllers*. Presented at the workshop on Applying Machine Learning in Practice at IMLC-95.

Schwefel, H.-P. (1965). *Kybernetische evolution als strategie der experimentellen forschung in der strömungstechnik*. Master's thesis, Technical University of Berlin.

Schwefel, H.-P., & Männer, R. (Eds.) (1991). *Parallel Problem Solving from Nature*, Volume 496 of *Lecture Notes in Computer Science*. Dortmund, Germany: Springer-Verlag.

Searson, D., Willis, M., & Montague, G. (1998). Chemical process controller design using genetic programming. See Koza, Banzhaf, Chellapilla, Deb, Dorigo, Fogel, Garzon, Goldberg, Iba, and Riolo (1998), pp. 359–364.

Sebald, A. V., & Fogel, L. J. (Eds.) (1994). *Proceedings of the Third Annual Conference on Evolutionary Programming*. San Diego, CA: World Scientific.

Sette, S., Boullart, L., & Van Langenhove, L. (1998). Using genetic algorithms to design a control strategy of an industrial process. *Control Engineering Practice*, *6*, 523–527.

Shim, H.-S., & Kim, J.-H. (1995). Robust control of non-holonomic wheeled mobile robot based on evolutionary programming for optimal motion. See IEEE (1995a), pp. 625–630.

Shimooka, H., & Fujimoto, Y. (2000). Generating robust control equations with genetic programming for control of a rolling inverted pendulum. See Whitley, Goldberg, Cantú-Paz, Spector, Parmee, and Beyer (2000), pp. 491–495.

Smith, S. (1995). An evolutionary program for a class of continuous optimal control problems. See IEEE (1995b), pp. 418–422.

Smith, S., & Stonier, R. (1996). Applying evolution program techniques to constrained continuous optimal control problems. See IEEE (1996), pp. 285–290.

Soucek, B., & the IRIS Group (Eds.) (1992). *Dynamic, Genetic, and Chaotic Programming: The Sixth Generation*. New York, NY: John Wiley & Sons, Inc.

Spector, L., Goodman, E. D., Wu, A., Langdon, W. B., Voigt, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M. H., & Burke, E. (Eds.) (2001). *Proceedings of the Genetic and Evolutionary Computation Conference 2001*. San Francisco, CA: Morgan Kaufmann Publishers.

Storn, R. (1996). Differential evolution design of an IIR-filter. See IEEE (1996), pp. 268–273.

Storn, R., & Price, K. (1996). Minimizing the real functions of the ICEC '96 contest by differential evolution. See IEEE (1996), pp. 842–844.

Tamaki, H., Sakakibara, K., Murao, H., & Kitamura, S. (2002). Modeling and meta-heuristic solution for flexible shop scheduling. See Camacho (2002).

Tanaka, K., & Hatanaka, S. (1995). Genetic tuning with concept of stabilizable chromosomes in nonlinear controller design. See IEEE (1995b), pp. 334–339.

Thierens, D., & Vercauteren, L. (1991). A topology exploiting genetic algorithm to control dynamic systems. See Schwefel and Männer (1991), pp. 104–108.

Thompson, A. (1995). Evolving electronic robot controllers that exploit hardware resources. In Morán, F., Moreno, A., Merelo, J. J., & Chacon, P. (Eds.), *Advances in Artificial Life: Proceedings of the 3rd European Conference on Artificial Life (ECAL95)*, Volume 929 of *Lecture Notes in Artificial Intelligence* (pp. 640–656). Granada, Spain: Springer-Verlag. Also available as University of Sussex School of Cognitive and Computing Sciences Technical Report CSRP368.

Topalov, A. V., Kim, K.-C., Kim, J.-H., & Lee, B.-K. (1996). Fast genetic on-line learning algorithm for neural network and its application to temperature control. See IEEE (1996), pp. 649–654.

Uchibe, E., Yanase, M., & Asada, M. (2001). Evolution for behavior selection accelerated by activation/termination constraints. See Spector, Goodman, Wu, Langdon, Voigt, Gen, Sen, Dorigo, Pezeshk, Garzon, and Burke (2001), pp. 1122–1129.

Urbančič, T., & Bratko, I. (1992). Knowledge acquisition for dynamic system control. See Soucek and the IRIS Group (1992) (Chapter 3, pp. 65–83).

Ursem, R. K. (2003). *Models for evolutionary algorithms and their applications in system identification and control optimization*. Unpublished doctoral dissertation, University of Aarhus, Dept. of Computer Science, Aarhus, Denmark.

Velasco, J. R. (1998). Genetic-based on-line learning for fuzzy process control. *International Journal of Intelligent Systems*, *13*, 891–903.

Velasco, J. R., & Magdalena, L. (1995). Genetic algorithms in fuzzy control systems. See Winter, Périaux, Galán, and Cuesta (1995) (Chapter 8, pp. 141–165).

Wang, P. P., Chen, S.-H., Cheng, H.-D., Gattiker, J. R., Georgiou, G. M., Laws, M., Lu, M., Romay, M. G., Scwartz, D. G., Shenoi, S., Shih, T. K., Wang, J. T. L., & Ying, H. (Eds.) (2000). *Proceedings of the Fifth Joint Conference on Information Sciences 2000*, Volume 1. Atlantic City, NJ: Elsevier Publishing Company, Inc.

Watson, J. B. (1994). Behavior-based control for autonomous robotics. See Sebald and Fogel (1994), pp. 185–190.

Whiteson, S., Kohl, N., Miikkulainen, R., & Stone, P. (2003). Evolving keepaway soccer players through task decomposition. In Cantú-Paz, E., Foster, J. A., Deb, K., Davis, D., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Standish, R., Kendall, G., Wilson, S., Harman, M., Wegener, J., Dasgupta, D., Potter, M. A., Schultz, A. C., Dowsland, K., Jonoska, N., & Miller, J. (Eds.), *Genetic and Evolutionary Computation – GECCO-2003, Part I*, Volume 2723 of *Lecture Notes in Computer Science* (pp. 356–368). Chicago, IL: Springer-Verlag.

Whitley, D. (1989). The GENITOR algorithm and selective pressure: Why rank-Based allocation of reproductive trials is best. See Schaffer (1989), pp. 116–121.

Whitley, D., Goldberg, D. E., Cantú-Paz, E., Spector, L., Parmee, L., & Beyer, H.-G. (Eds.) (2000). *Proceedings of the Genetic and Evolutionary Computation Conference 2000*. Las Vegas, NV: Morgan Kaufmann Publishers.

Winter, G., Périaux, J., Galán, M., & Cuesta, P. (Eds.) (1995). *Genetic Algorithms in Engineering and Computer Science*. West Sussex, England: John Wiley & Sons, Ltd.

Yamazaki, K., Kundu, S., & Hamano, M. (1998). Genetic programming based learning of control rules for variable geometry structures. See Koza, Banzhaf, Chellapilla, Deb, Dorigo, Fogel, Garzon, Goldberg, Iba, and Riolo (1998), pp. 412–415.

Yoon, J. Y., Hwang, G.-H., & Park, J. H. (1998). A genetic algorithm approach to design an optimal fuzzy controller for rectifier current control of HVDC system. See IEEE (1998), pp. 404–409.

Zhao, Y., Collins, E. G., & Dunlap, D. (2003). Design of genetic fuzzy parallel parking control systems. See American Automatic Control Council (2003), pp. 4107–4112.

Zitar, R. A. A., & Hassoun, M. H. (1993). Regulator control via genetic search assisted reinforcement. See Forrest (1993), pp. 254–262.

Zitzler, E., Laumanns, M., & Thiele, L. (2001, May). *Spea2: Improving the strength pareto evolutionary algorithm* (TIK-Report No. 103). Zurich, Switzerland: Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH).

Zitzler, E., & Thiele, L. (1998, May). *An evolutionary algorithm for multiobjective optimization: The strength pareto approach* (TIK-Report No. 43). Zurich, Switzerland: Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH).

Zitzler, E., & Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, *3*(4), 257–271.